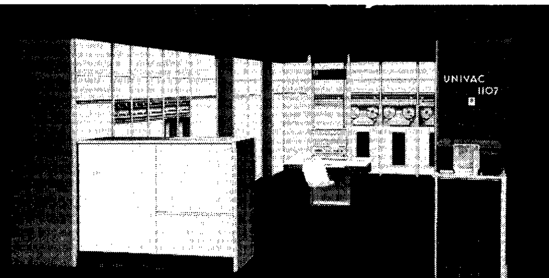


UNIVAC® 1107

TECHNICAL BULLETIN

UNIVAC 1107

CENTRAL COMPUTER

November, 1961

CONTENTS

1. UNIVAC 1107 THIN-FILM MEMORY COMPUTER

GENERAL DESCRIPTION

The UNIVAC® 1107 Thin-Film Memory Computer signals the arrival of a third generation of computers since it marks the first time thin-film memory is used in a commercially available data-processing system. Thin-film memory – the most significant technological achievement since solid-state circuitry – brings to commercial and scientific computer users data control and storage techniques never before available.

Designed and developed as a solid-state, general-purpose system, the UNIVAC 1107 Computer utilizes advanced data-processing methods. Its concept of design centralizes the many controls – necessary for high efficiency input and output, concurrent computation, and internal transmission – within the thin-film memory, the "heart" of the system.

As a direct result of its logical design, the UNIVAC 1107 Computer can reliably and economically process a wide range of applications in either an on-line or an off-line mode. Equally important is the rate of speed at which these applications can be processed: internal speeds of the UNIVAC 1107 System are measured in nanoseconds – billionths of a second. Accordingly, the system is particularly well equipped to handle real-time applications.

UNIVAC thin-film, also known as magnetic film, is manufactured by deposition of vaporized magnetic alloys on thin planes of glass under the influence of a strong magnetic field. Because these deposits are made in extremely thin layers, the direction of their magnetic field can be switched in an interval of several nanoseconds. This feature allows information to be stored or retrieved at extremely high rates of speed. Immediate benefits include substantial savings in processing time, reduced power requirements, and miniaturized storage units.

Basically, magnetic-film memory consists of an array of minute circular metallic elements, several millionths of an inch thick, deposited on planes (substrates) of glass. Functionally, each metallic deposit (or "dot") can be compared to the ferrite core employed in conventional storage units. Thirty-six "dots" are assigned to each of the 128 words in the thin-film or control memory.

Instead of wires physically threading ferrite cores, the circuitry for magnetic film is printed on MYLAR* tape, and then wrapped around glass substrates. Figure 1-1 depicts a glass substrate after deposition of the metallic alloy.

Employed primarily in a control capacity, magnetic-film memory provides multiple accumulators, index registers, control-registers, and input-output registers. As a result of this arrangement, intricate input-output, arithmetic, and housekeeping operations – which formerly required extensive data manipulation – have now become little more than routine programming functions.

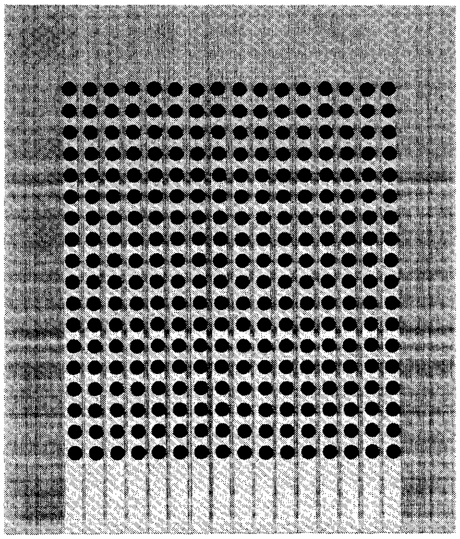


Figure 1-1. Substrate of Magnetic Film.

**Registered trademark of the E.I. duPont de Nemours and Company for its polyester film.*

FEATURES

Among the more prominent features of the UNIVAC 1107 Data-Processing System are:

- A magnetic-film memory – the most advanced data storage device on the market today.
- A ferrite-core memory for instructions and operands, available in capacities of 16,384 words in one bank or 16,384 words and multiples thereof (up to 65,536 words) in two separately accessed banks.
- Six hundred and sixty-seven nanosecond (0.667 microsecond) cycle time for film memory, complemented by an effective 2-microsecond cycle time for core memory (overlapping of two banks).
- Sixteen bidirectional input-output channels, capable of concurrent input-output transmissions at a maximum rate of 250,000 words (1,500,000 characters) per second.
- Automatic programming, including ALGOL, COBOL, FORTRAN, simulators and assemblers.
- An executive system for integrating the subroutines required in the processing of multiple programs.
- A highly versatile instruction word that provides for indexing, indirect addressing, automatic incrementation of the modifier, and partial word transmission, as well as specification of both an operand and an arithmetic register.
- A repertoire of instructions that frequently combines two or more data-processing opertions in a single command.

PERIPHERAL EQUIPMENT

The input-output section of the UNIVAC 1107 Computer System accommodates many different types of peripheral equipment. Some external units, such as magnetic drum and tape units, may be used to provide auxiliary storage. Other devices may serve as input-output equipment; these would include card and tape units, printers, and document-sensing devices. Additional special peripheral equipment can provide information links to other systems.

Standard on-line peripheral equipment for the UNIVAC 1107 System consists of:

Magnetic Drum Storage Systems (FH880 Drums)

Magnetic Tape Units:

UNISERVO IIA Units (Remington Rand UNIVAC format)

UNISERVO IIA Units (IBM format)

UNISERVO III Units

Card Readers (80-column or 90-column)

Card Punches (80-column or 90-column)

High-Speed Printer

Paper Tape Reader

Paper Tape Punch

Conventional off-line operations, such as card-to-tape conversions, can be performed on-line with negligible interruption of running programs. In this type of operation, data flows to and from an assigned memory area. In effect, this memory area serves as a data transfer buffer, functioning independently of the main program.

The capacity of the UNIVAC 1107 System to absorb many off-line operations eliminates the need for special equipment. Appreciable savings in floor space, power requirements, and rental costs are then realized.

2. CENTRAL COMPUTER

The Central Computer in the UNIVAC 1107 System consists of four major sections: storage, control, arithmetic, and input-output.

STORAGE

Regardless of the selected core memory capacity, each UNIVAC 1107 System is equipped with a separate magnetic-film memory. Consequently, the storage section of the Central Computeŕ encompasses both a magnetic-film memory and a core memory, along with their respective address, transfer, and control circuits.

Magnetic-Film Memory

Magnetic film in the UNIVAC 1107 System provides a 128-word control memory. Each word, is capable of storing 36 bits of information. The film array is such that word selection determines which 36 bits are to be accessed. Operating in the parallel mode, read access time for any film-memory address is 167 nanoseconds (0.167 microsecond); complete cycle time is 667 nanoseconds.

The magnetic-film memory is the most frequently used area in the entire UNIVAC 1107 Data-Processing System. As a general rule, in the time it takes to make a single reference to core memory, film memory will have been referenced *three* times. Carried a step further, approximately 1.5 million references per second can be made to film memory.*

Core Memory

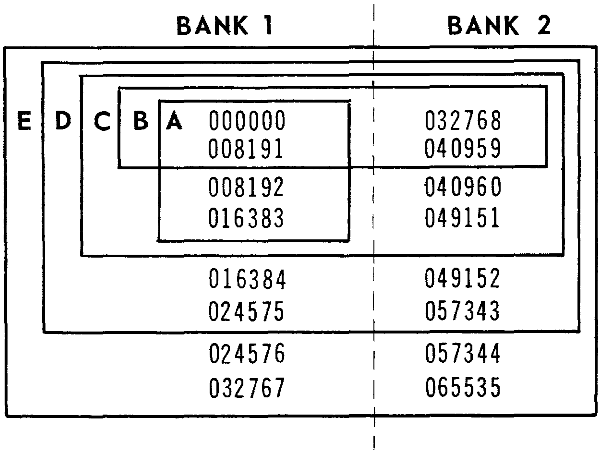
Core storage in the UNIVAC 1107 System consists of small doughnut-shaped magnetized cores of ferrite material. Depending upon its direction of magnetic orientation, each core (similar to the metallic "dot" in film memory) is capable of representing one of two stable states: on or off (1 or 0).

The cores themselves are arranged in planes. Wires thread the planes in a pattern similar to that of the vertical and horizontal coordinate lines on a map. The intersection of two wires determines a specific core. Data stored in core memory is accessed via word selection and read in the parallel mode.

UNIVAC 1107 core memory is available in options of 16,384 words in one bank; or, 16,384, 32,768, 49,152, or 65,536 words in two banks. Read access time for any core-memory address is 1.8 microseconds; complete cycle time is 4.0 microseconds.

In a two-bank installation, regardless of the selected memory capacity, the full range of lower-order addresses (0 through 32,767) apply to bank one, while the full range of higher-order addresses (32,768 through 65,535) apply to bank two.

To illustrate this principle, assume a two-bank installation has a total storage capacity of 32,768 words. As shown in Figure 2-1 (Option C), the addresses available to the programmer are 0 through 16,383 in bank 1 and 32,768 through 49,151 in bank 2, for a total of 32,768 locations. Note that bank 1 does not end at address 16,383 and bank 2 begin with address 16,384. Instead, each bank has the full complement of addresses. In this manner, the system lends itself to future expansion.



OPTION	CAPACITY/WORDS	BANK
A	16,384	1
B	16,384	1, 2
C	32,768	1, 2
D	49,152	1, 2
E	65,536	1, 2

Figure 2-1. Core Storage Options.

In this case references to core memory are overlapped to provide an efficient communications rate of 500,000 words per second.

BANK 1			BANK 2		
DECIMAL	OCTAL	BINARY	DECIMAL	OCTAL	BINARY
000000	000000	0 000 000 000 000 000	032768	100000	1 000 000 000 000 000
		⋮			⋮
008191	017777	0 001 111 111 111 111	040959	117777	1 001 111 111 111 111
008192	020000	0 010 000 000 000 000	040960	120000	1 010 000 000 000 000
		⋮			⋮
016383	037777	0 011 111 111 111 111	049151	137777	1 011 111 111 111 111
016384	040000	0 100 000 000 000 000	049152	140000	1 100 000 000 000 000
		⋮			⋮
024575	057777	0 101 111 111 111 111	057343	157777	1 101 111 111 111 111
024576	060000	0 011 000 000 000 000	057344	160000	1 011 000 000 000 000
		⋮			⋮
032767	077777	0 111 111 111 111 111	065535	177777	1 111 111 111 111 111

ACTUAL ADDRESS ACCESSED

PROGRAM REFERENCED ADDRESS

Figure 2-2. Decimal, Octal, and Binary Values of Core Storage Addresses

In a system that uses less than maximum storage, an address that exceeds the capacity of a selected bank will automatically reference an address in that same bank with fewer significant bits. For example, if memory capacity is 32,768 words and the programmer inadvertently references address 65,535, the last address for maximum storage, the program will automatically access location 49,151, the highest actual address in bank 2.

The principal advantage of a two-bank installation is that by simply storing data in one bank and instructions in the other, core-memory references in consecutive instructions can be overlapped. Under this arrangement, the cores that contained the current instruction's operand can be read, while the cores in the alternate bank, containing the next instruction, are being read. The net result is an *effective* cycle time of 2.0 microseconds (see Figure 2-3).

Storage Allocation

The addresses of the 128 locations in film memory are identical to those of the first 128 locations in core memory. Distinction between memory units is based on whether the address is specified by the program-address register *(P)* or a designator in the instruction in the Program Control Register (PCR). If the address of a location that can be found in both film memory and core memory is contained in *P*, an instruction word is being accessed. Consequently, program control will automatically reference the appropriate location in core memory. Conversely, if the address is specified via the *PCR*, a data word, a constant, or a control word, is stipulated. In this case, program control automatically references film memory.

	Data	Instruction
Cycle Time Microseconds	Bank 1	Bank 2
2		READ
2		RESTORE
2	READ	READ
2	RESTORE	RESTORE
2	READ	READ

Overlap. Effective Cycle Time of 2 μs

Figure 2–3. Overlapped Core-Memory References

Magnetic-Film Memory

The 128 locations in magnetic-film memory are reserved for data words, constants and control words. These locations can only be accessed by load, add, mask, and similar instructions; that is, an instruction that designates an internal operation. For example, when a programmer specifies that the contents of location 20 are to be stored in location 115, the contents of *film-memory* location 20 will be transferred to *film-memory* location 115.

A particular location in film-memory is accessed via designators in the instruction in the *PCR*. The contents of the specified location are then transferred either to the arithmetic section or to another memory location.

An input-output instruction, that is, one that specifies transfers to or from peripheral equipment, will only reference core memory. This means that whenever the contents of a film-memory location are to serve as output, they must first be transferred to a core-memory address. Similarly, input data that is to be operated on arithmetically must be transferred first to core memory, and then, to film memory.

In this respect, the UNIVAC 1107 incorporates a unique safety feature. In refusing input-output instructions direct access to magnetic-film memory, the system precludes the possibility of a programmer inadvertently overlaying input data on control-memory data essential to computation.

As employed in the Central Computer, magnetic film supports a comprehensive data-processing network. For example, many of the system's advanced processing and input-output options result from the 63 magnetic-film locations that provide:

Index registers

Arithmetic registers

Input-output access control registers

Temporary program address register

Real-time clock

Mask register

Repeat count register

The programmer is free to use the remaining locations as auxiliary storage for data and constants.

Core Memory

Just as film memory is reserved for data words and is protected from external operations, the first 128 locations in core memory are reserved for instruction words and are fully protected from all internal write operations. Because of this logical design, these locations are particularly well suited for a bootstrap routine.

Individual instructions or a bootstrap routine may be loaded into the first 128 locations* in core memory only by means of input peripheral equip-

* *The bootstrap routine may utilize up to 224 locations in core memory.*

CONTROL MEMORY (Thin-Film)		CORE MEMORY			
		BANK 1		BANK 2	
128 36-bit Words 0.667μs Cycle Time		8,192, 16,384 or 32,768 36-bit Words 2μs Cycle Time (effective)		8,192, 16,384 or 32,768 36-bit Words 2μs Cycle Time (effective)	
Z_0 (I/O Register)	S_0 (Storage Address Register)	S_1 (Storage Address Register)	Z_1 (I/O Register)	S_2 (Storage Address Register)	Z_2 (I/O Register)

Figure 2-4. UNIVAC 1107 Thin-Film Memory Computer Storage

ment. Once stored, the instructions can be altered only by reading new instructions, via peripheral equipment, into the same locations. Behind this stipulation lies the general rule that whenever the designators in an instruction specify an address that may be found in both film memory and core memory, the program will reference film memory.

Instructions stored in the first 128 core-memory locations are accessed via *P*, the program address register. The contents of the specified location are then transferred to *PCR*, the program control register, for execution. *Note* that entry into *PCR* can only be gained from core memory.

The next 75 core-memory locations (addresses 128 through 202) are reserved for interrupts and the external status word. The remaining locations in core memory (addresses 203 through 65,535 when maximum capacity is used) may be employed as the programmer desires.

CONTROL

The control section of the Central Computer comprises the program address register, the program control register, the storage class control decoding unit and the indexing unit. In addition, this section includes the circuits which supply the control signals necessary to synchronize the execution of instructions.

The program address register, *P*, normally contains the address of the next instruction, except during a repeated sequence operation when it contains the remaining number of times the instruction is to be executed. The program control register, *PCR*, contains the instruction currently being executed. The storage class control decoding unit, *SCC*, decodes the effective operand address for subsequent referencing to magnetic-film memory or core-memory bank 1 or bank 2.

Indexing Unit

The indexing unit, containing an adder and sensing circuits, is shared by both program control and input-output control. Program control uses the indexing unit to: advance the *P*-register by 1 each time an instruction is executed (provision is thus made for sequential execution of instructions); to count down and control repeated sequences; and to perform address modification, and incrementation.

The indexing unit performs address modification as 18-bit one's complement addition. Because the maximum operand address utilizes 16 bits, two binary 0's are placed to the immediate left of the operand address. After modification, the two most significant bits in the effective operand address are dropped and the 16-bit address is transferred to *SCC*.

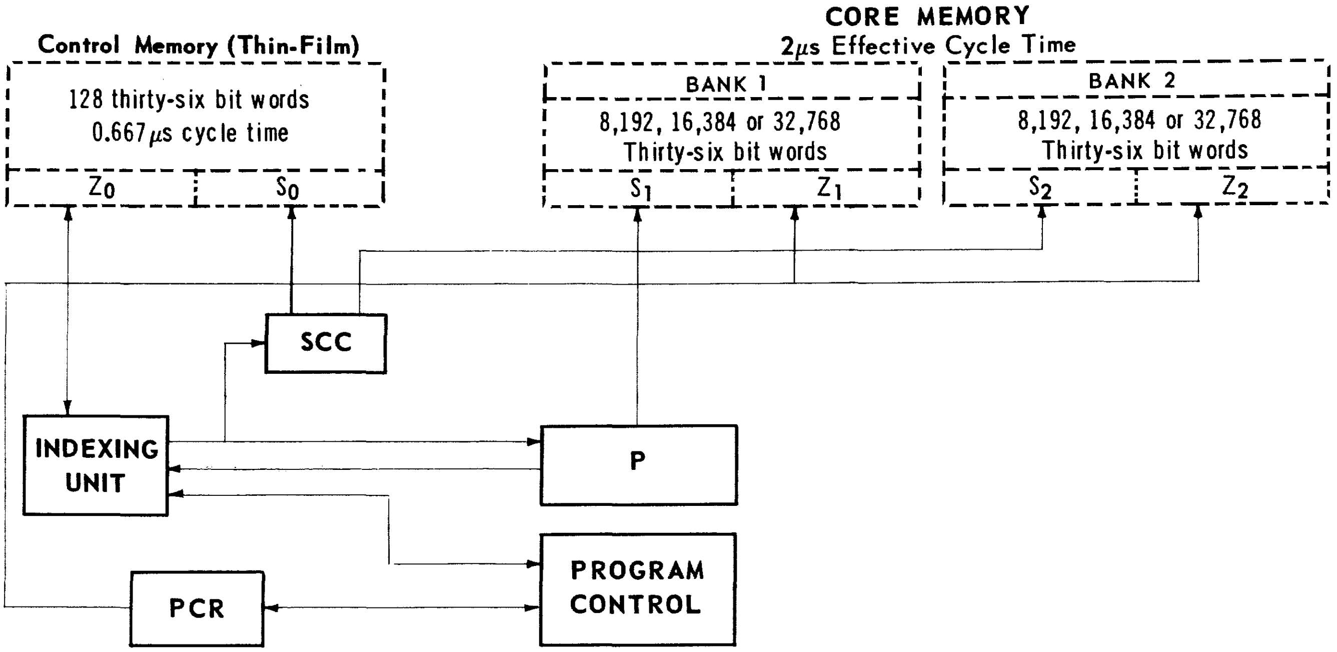


Figure 2-5. Control Paths and Units

Input-output control uses the indexing unit to specify both the number of words to be transferred and the locations to or from which data will move.

Interrupts

Interrupts are special control signals which divert the attention of the computer from the main program to a particular event or set of circumstances. In the UNIVAC 1107 System, provision is made for several classes of interrupts.

There is an external interrupt for each of the 16 input channels. These interrupts enable peripheral equipment to request access to the Computer. There are internal interrupts corresponding to each of the 16 input access-control words, 16 output access-control words, and the 16 external function words. An internal interrupt is also provided for the real-time clock.

An additional external interrupt is available for real-time system synchronization. This interrupt is independent of the input-output channels. It accepts signals of any frequency from an external generator which may be a supplementary *real-time clock* for the Central Computer or the *master clock* for a multiple-computer installation.

In the UNIVAC 1107 System, interrupts need not be tested to reveal their source. Instead, each interrupt is associated with a *fixed* address which automatically provides entry into a subroutine corresponding to the event or circumstances that caused the interrupt.

Initial Load Operation (Automatic Bootstrap)

An initial load operation is provided for initial loading of programs and for program restoration. The initial load operation will read a maximum of 224 words from peripheral equipment into the first 224 locations in core memory. Upon termination of the reading, program control is transferred to the program contained within these 224 words. The initial load operation may be initiated manually or by program control.

ARITHMETIC

The arithmetic section of the Central Computer includes threshold sensing circuits, counters, arithmetic sequence control circuits, a shift matrix, temporary storage registers, and an adder.

The threshold sensing circuits determine the equality and relative magnitude of the contents of specified registers. The counters are employed during multiply and divide operations. Sequence control circuits govern the execution of add, subtract, multiply, divide, shift, and test-relative-

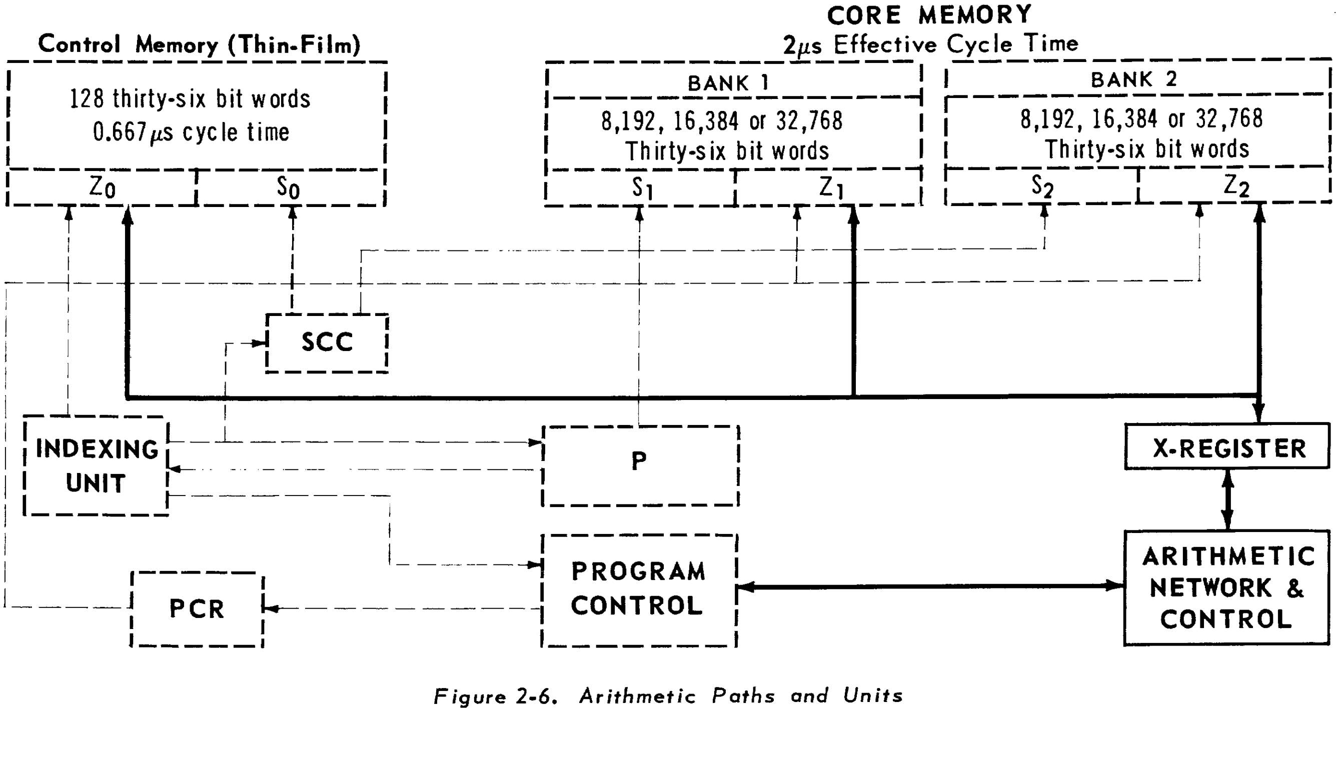


Figure 2-6. Arithmetic Paths and Units

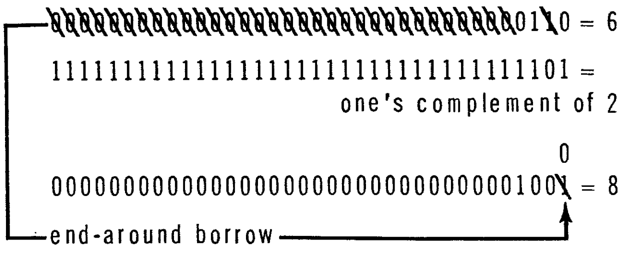
magnitude instructions. The shift matrix shifts data from 0 through 36-bit positions in a shift operation.

During the actual execution of an arithmetic instruction, temporary storage registers within the arithmetic section itself are employed. The Central Computer first determines that the arithmetic section will be utilized in a given operation. Data is then transferred automatically, via the *X*-register*, to a temporary storage register of the arithmetic section. The *X*-register and the temporary storage registers cannot be addressed by the programmer.

Adder

The adder in the UNIVAC 1107 System is a 36-bit one's complement subtractive adder (mod $2^{36} - 1$). Additions are performed in the following manner:

> Assume the value 2 is to be added to the value 6. In core storage, the binary equivalents of these values are:
>
> ```
> 000000000000000000000000000000000110 = 6
> 000000000000000000000000000000000010 = 2
> ```
>
> In executing the instruction, the adder first complements the value 2:
>
> ```
> 111111111111111111111111111111111101 =
>                  one's complement of 2
> ```
>
> Next, the adder subtracts the one's complement of 2 from the value 6. The subtraction itself involves an "end-around borrow," whereby the process of borrowing from the digit to the left may carry from the leftmost digit in the minuend (value 6) to the rightmost digit in the remainder. It will continue moving to the left in the remainder until the borrow is satisfied:



In the example, the binary 1 in digit position 3* in the subtrahend cannot be subtracted from the binary 0 in the corresponding minuend position. If the subtraction is to continue, a binary 1 must be borrowed from a digit to the left in the minuend. However, digit positions 4 through 35 all contain binary 0's. At this point an end-around borrow occurs; that is, the needed binary digit is taken from the remainder. As it happens, the first bit position in the remainder contains the binary 1 needed to satisfy the borrow.

After the end-around borrow, computation adheres to the rules of binary subtraction. The remainder is the sum of the values 6 and 2.

Overflow and Carry Designators

Associated with the adder are two special designators: the overflow designator and the carry designator. Eight instructions affect the two designators: the four basic add instructions and the four basic subtract instructions – operation codes (in octal notation) 14 through 21, 24 and 25.

Upon execution of one of the eight instructions, both designators are cleared. After addition has been performed, the designators remain in their respective states (set or clear) until another one of the eight instructions is given. Both designators are set in time to be tested immediately after the affecting instruction.

The overflow designator is set upon generation of a significant bit in the sign position. This condition can only arise when the values that are added have like signs. Specifically, a positive result of two negative quantities will set the overflow designator, as will a negative result of two positive values.

The carry designator is set whenever an end-around carry (no borrow) is generated. The condition of the carry designator is determined by the following rules:

VALUES	POSITIVE RESULT	NEGATIVE RESULT
A positive and U negative	Set	Clear
A negative and U positive	Set	Clear
A negative and U negative	Set	Set
A positive and U positive	Clear	Clear

**X is the 36-bit exchange register providing entrance and exit to the arithmetic section.*

**Reading from right to left the bit positions are numbered 0 through 35.*

The following additions will always set the carry designator:

1. *Any number added to its complement*
2. *All 0's added to all 1's.*
3. *Any number added to all 1's.*
4. *All 1's added to all 1's.*

Arithmetic Registers

Sixteen arithmetic or *A*-registers, directly addressable by the programmer, are available for storing operands and results of arithmetic operations. These 16 registers are not to be confused with the non-addressable temporary storage registers within the arithmetic section itself.

As previously pointed out, during actual computation temporary storage registers in the arithmetic unit are utilized. However, these registers are not capable of retaining initial data or final results from one instruction to another. Consequently, all such information is transferred automatically to the *A*-registers specified in the instructions. The 16 *A*-registers, then, function as accumulators.

Partial Word Transfers

Word transmissions between the arithmetic section and core memory can be directly segmented into halves, thirds, or sixths. This flexibility allows the Central Computer to operate upon one of eleven possible portions of a word or the entire 36-bit word itself, as shown in Figure 2-7. The selected data in a partial word transfer from memory is shifted automatically to lower-order positions in the arithmetic section. By means of this feature, computation can be performed immediately after the partial words have been transferred, without first calling for such housekeeping instructions as shifts。

Along with partial word transfers, special add and subtract instructions are available to the programmer. Upon execution of one of these instructions, parallel addition or subtraction of two or three fields within a single data word is performed.

Floating-Point Arithmetic

In the UNIVAC 1107 System, floating-point arithmetic has been made a hardware, rather than a software or programming, function. Seven instructions are devoted exclusively to floating-point arithmetic. Addition, subtraction, and multiplication always store a 2-word result. Both results contain their appropriate characteristics. Division produces a quotient and a remainder, both of which are in the floating-point format. The 2-word results of these floating-point instructions lend themselves to programmed double-precision arithmetic.

INPUT-OUTPUT

The input-output section of the Central Computer provides the data paths and control circuits necessary for direct communication between core memory and peripheral equipment. Data transfers may be scheduled over a maximum of 16 bidirectional input-output channels. When 16 channels are operating concurrently, word transfers can be multiplexed to provide a maximum communication rate of 250,000 words (1,500,000 characters) per second. Of course, such high-speed input-output data transfer rates are rarely maintained for more than brief periods.

The main computer program establishes the initial communications path between core memory and the peripheral equipment. From this point on, individual word transfers are governed by input-output access-control circuits. These circuits monitor the number of words to be transferred and specify the core-memory addresses to and from which data are transmitted. In this way, the access-control circuits allow the Central Computer to resume execution of the main program.

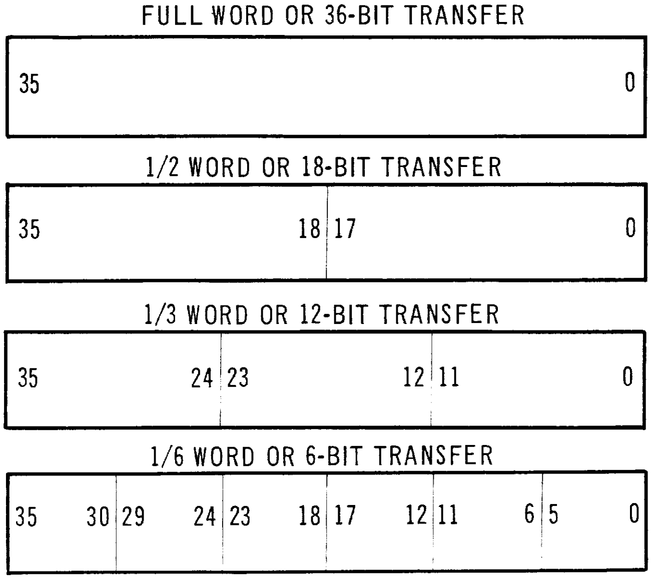

FULL WORD OR 36-BIT TRANSFER
35
0
1/2 WORD OR 18-BIT TRANSFER
35
18
17
0
1/3 WORD OR 12-BIT TRANSFER
35
24
23
12
11
0
1/6 WORD OR 6-BIT TRANSFER
35
30
29
24
23
18
17
12
11
6
5
0

Figure 2-7. Partial Word Transfers

3. DATA, CONTROL, AND INSTRUCTION WORDS

DATA REPRESENTATION

Internal operations in the UNIVAC 1107 System are performed in the parallel binary mode. Since the machine language is binary, data, control, and instruction words must be expressed in pure binary form. However, for convenience in programming, as well as in monitoring internal operations, octal notation can be used.

The basic data word in the UNIVAC 1107 System, as shown in Figure 3-1, utilizes 36 binary digit positions. Position 35 contains the sign bit, bit position 34 is the most significant, and bit position 0 is the least significant.

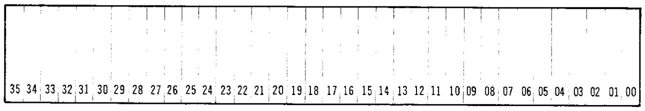


Figure 3-1. Basic Internal Data Word.

With one position reserved for the sign, a total of 35 binary digit positions may be used to represent a given quantity. The largest number that can be accommodated in the UNIVAC 1107 System (exclusive of floating-point and double-precision arithmetic) is $2^{35} - 1$ or 34,359,738,367.

Positive binary numbers are obtained in the following manner. The absolute value of the desired number is placed in the low-order positions of a given register. A 0 is placed in bit position 35 and extended right until a binary 1 is reached.

Example

+ 9 = 000 000 000 000 000 000 000 000 000 000 001 001

Negative binary numbers, on the other hand, are arrived at by complementing (substituting a binary 1 for each binary 0 and a binary 0 for each binary 1) the positive binary configuration of the desired negative value. Applying this rule, a negative 9 is obtained by complementing the binary representation of a positive 9.

Example

− 9 = 111 111 111 111 111 111 111 111 111 111 110 110

NOTE: Positive numbers are characterized by a 0 in bit position 35 and negative numbers by a 1 in bit position 35. Also, in positive numbers the first significant bit position contains a 1 and in negative numbers it contains a 0.

CONTROL WORDS AND CONTROL REGISTERS

Special 36-bit control words are associated with several types of film-memory registers. Data transferred to these registers should adhere to the format of the corresponding control word. Data that is to enter a register which is not associated with a special control word is transferred in the format of the basic data word.

Index Registers

Fifteen 36-bit registers are available in thin-film memory for index register modification and index counts. Index register word format is as follows: The right half (*Q*-portion) of the index register word stores the modifier which may be up to 18-bits (including sign) in length; the left half (the Δ portion) of the word stores an increment which can be up to 18-bits (including sign) in size. The index register word format is shown in Figure 3-2.

35*	Δ	18	17*	Q	00

*Sign Positions

Figure 3-2. Index Register Word.

When an indexing operation is indicated, the appropriate modifier is applied to the current instruction's base execution address. The result is the effective operand address (before indirect addressing, if specified). Then, depending upon the value of a special designator in each instruction, the increment is applied to the modifier. In this way, provision is made for varying the extent of address modification in subsequent indexing operations.

The leftmost bit in both the modifier and the increment (or decrement) portions of the index register word specifies these quantities as positive or negative.

Arithmetic Registers

Sixteen magnetic-film locations provide interim storage for arithmetic operands and results. Because four of these locations overlap addresses assigned to index registers (Table 1), the Central Computer in the UNIVAC 1107 System is capable of performing highly sophisticated address modification. For example, in a table look-up application, the results of a given calculation can immediately be applied, as a modifier, to a base address.

	DECIMAL ADDRESS	OCTAL ADDRESS	FUNCTION	
CONTROL MEMORY	00000	000000	Unassigned (depends on operation)	
	00001-00015	000001-000017	Index Registers (15)	
	00012-00027	000014-000033	Arithmetic Registers (16)*	
	00028-00031	000034-000037	Unassigned	
	00032-00047	000040-000057	Input Access-Control Words (16)	
	00048-00063	000060-000077	Output Access-Control Words (16)	
	00064	000100	Real-Time Clock	
	00065	000101	Repeat Counter	
	00066	000102	M Register	R Registers
	00067	000103	T-Register (temporary storage for P)	
	00068-00079	000104-000117	Additional Special Registers	
	00080-00127	000120-000177	Unassigned	
CORE MEMORY	00000-00127	000000-000177	Unassigned **	
	00128-00143	000200-000217	External Request Interrupts (16)	
	00144-00159	000220-000237	Input Data Termination Interrupts (16)	
	00160-00175	000240-000257	Output Data Termination Interrupts (16)	
	00176-00191	000260-000277	Function Termination Interrupts (16)	
	00192-00199	000300-000307	Error Interrupts (8)	
	00200	000310	Real-Time Clock Interrupt	
	00201	000311	External Status Word	
	00202	000312	External Synchronization Interrupt	
	00203-65535	000313-177777	Unassigned Core-Memory Addresses	

*Memory addresses 000014-000017 are also addressable as index registers
** Normally reserved for Bootstrap Routine.

Table 1. Storage Allocation of Film and Core Storage Locations

The format of data that is to be loaded into an arithmetic or *A*-register is contingent upon the type of arithmetic operation to be performed. In the case of fixed-point arithmetic, operands need only conform with the format of the basic data word. Floating-point arithmetic, however, requires a word format of its own. The *floating-point word,* depicted in Figure 3-3, contains a 27-bit mantissa, an 8-bit characteristic, and a sign bit.

Sign	Characteristic*		Mantissa	
35	34	27	26	00

* Biased by 128 (200 octal)

Figure 3-3. Floating Point Word

As previously mentioned, the *A*-registers function as 16 accumulators; that is, they retain the results of computation from one instruction to another.

R-Registers

Sixteen film-memory locations are designated as "*R*-registers." Twelve of these registers may be used in any way the programmer desires except, that they, as well as all other film-memory locations, cannot be employed for storing instructions. As shown in Table 1, the remaining four *R*-registers are assigned the following specific functions:

Real-Time Clock

One of the four assigned *R*-registers serves as the real-time clock. Every millisecond (the exact timing is 2^{-10} seconds), the 36-bit number contained in th *Figure 3-5. T-Register Word.* by 1. When the count reaches 0, an internal interrupt occurs which causes the main program to jump to address 200 (octal 310). Therefore, the programmer must either load the real-time clock register or provide for recovery from the automatic interrupts generated by the clock every 2^{-10} seconds.

The real-time clock, along with the other *R*-registers, the index registers, and the arithmetic registers, may be referenced directly either in the operand portion of an instruction (*u* address) or in the arithmetic register designator (*a* address). (This is known as 2-address accessibility, and it is explained further on page .) In respect to the real-time clock, two-address accessibility simplifies the setting and subsequent reading of the count. The real-time clock is not associated with a special control word.

Repeat Count

The second of the four assigned *R*-registers (referred to as the "*K*-register") provides the repeat count during the execution of an instruction in the repeat sequence mode. Data that is to enter this register should be in the format of the repeat count word, as shown in Figure 3-4.

unassigned		k	
35	18	17	00

Figure 3-4. Repeat Count Word.

Initially, the *k* portion of this control word contains the total number of times a particular instruction is to be executed. Then, during the repeat operation itself, *k* is reduced by 1 each time the instruction is executed. Provision is thus made for a "running" count of the number of execution times remaining in sequence. When *k* reaches 0, the repeat operation is terminated.

In certain applications it may be necessary to retain the *initial* repeat count. To meet this program requirement, load the repeat count (the total number of times an instruction is to be executed) into the unassigned left half of the repeat count word as well as into the *k* portion.

Mask Register

The third assigned *R*-register contains the mask (bit pattern) used in certain logical and test instructions. Data that is to enter the mask or *M*-register, is in the format of the basic data word.

Temporary Program Address Register

The fourth *R*-register assigned a specific function is employed as a temporary storage register *(T*-register) for the address of the next instruction. Utilized only during a repeat operation, the program address is stored in the next instruction portion of the *T*-register word. The format of this particular control word is presented in Figure 3-5.

unassigned		Next Instruction	
35	18	17	00

Figure 3-5. T–Register Word

Input-Output Access Control Registers

Thirty-two locations in film memory are used to maintain control over data transfers between the

Central Computer and peripheral equipment. Input-output access control words are associated with this group of registers. These words, along with the function words necessary to initiate input-output data transfers, are discussed in the UNIVAC 1107 Input-Output Manual. At this point, it is sufficient to note that thirty-two film-memory locations (addresses 32 – 63, Table 1), are reserved for this purpose.

INSTRUCTION WORD

The UNIVAC 1107 Thin-Film Memory Computer is controlled by a program of instructions stored in memory. Each instruction consists of various parts called designators. These designators are identified by letters, as shown in Figure 3-6.

f	j	a	b	h	i	u
35 30	29 26	25 22	21 18	17	16	15 0

f (6 bits) – Operation Code
j (4 bits) – Operand Instruction or Minor Operation Code*
a (4 bits) – A, B, or R-register, or Input-Output Channel Designator*
b (4 bits) – B-Register Designator
h (1 bit) – Incrementation Designator
i (1 bit) – Indirect Addressing Designator
u (16 bits) – Base Operand Address

* Instruction determines usage.

Figure 3-6. Basic Instruction Word.

Operation Code, f (6-Bits)

The operation code or *f* designator, composed of the leftmost six bit positions in the instruction word, stipulates the particular operation that is to be performed. (In certain instructions, when the normal meaning of the *j* designator is not applicable, the operation code may be expanded to include the ten leftmost bits in the instruction word.) Invalid *f* (or *f* and *j*) values are fault conditions which cause an error interrupt to occur. In this event, the main program jumps to a fixed memory address containing the entrance to an appropriate error subroutine.

Operand Interpretation, j (4-Bits)

Normally, the *j* designator determines whether an entire data word or only a part of it is to be transferred to or from the arithmetic section. As previously mentioned, in certain instructions *j* serves as a minor operation code rather than as a partial word determinant.

In the case of partial transfers, *j* stipulates which portion of a word (half, third or sixth) is to be transferred. Figure 3-7 shows the *j* values and corresponding word portion transfers to the *X*-register.

When *j* equals 16 or 17 (octal), the effective operand is taken directly from the instruction word, itself, instead of calling for an operand from memory.

In data transfers to the arithmetic section, when *j* equals 3 through 7 or 17, the sign of the operand, which is the MSB of the partial word, is extended to the high-order positions in the arithmetic section. Figure 3-7 shows that thirds are always extended, sixths are never extended, and extension is optional with half words. Figure 3-8 shows the *j* values and word portion transfers from the *X*-register to the core memory input-output registers Z_1 and Z_2. A *j* of 16 or 17 (octal) inhibits the data transfer.

A-Register Designator, a (4-Bits)

The type of instruction that is to be executed determines the specific usage of the 4-bit *a* designator.

In arithmetic instructions, *a* specifies one of sixteen arithmetic registers. In a few instructions, such as *Block Transfer; Load Ba Modifier Only;* and *Test Modifier*, the *a* designator specifies one of sixteen index registers. Input-output instructions, on the other hand, use *a* to stipulate which communications channel and access control word is to be used. In some instructions, the *a* designator specifies an *R*-register, using the notation R_a. In the *Index Jump* instruction, *a* and *j* combine to specify any desired control-memory location.

B-Register Designator, b (4-Bits)

The 4-bit *b* designator determines which of the fifteen index registers, if any, is to modify the instruction's operand address. When *b* equals 0, address modification is inhibited. (Index register 0 can only be addressed by the *a* designator.)

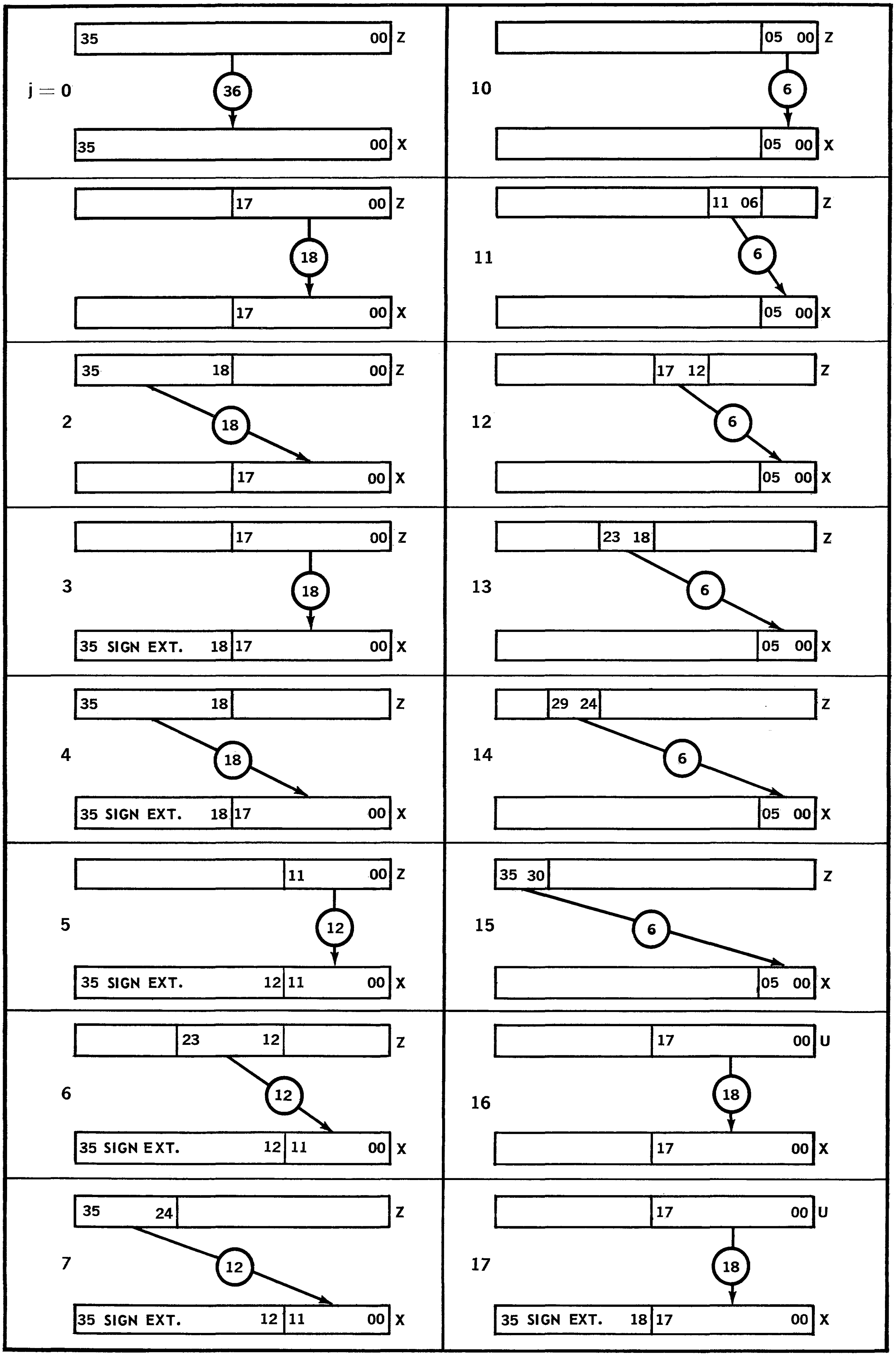


Figure 3-7. Data Paths to Arithmetic Section.

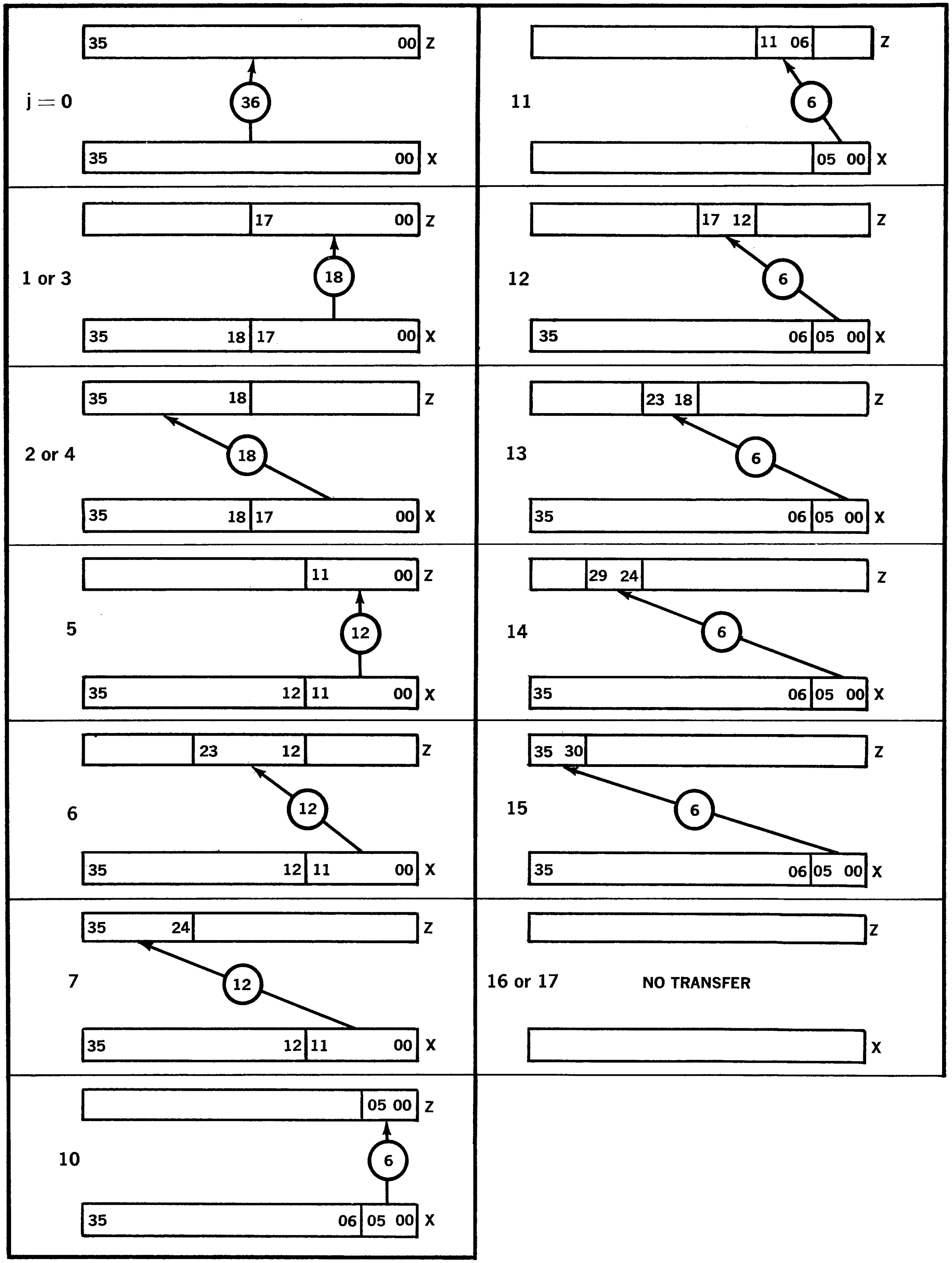


Figure 3-8. Data Paths from Arithmetic Section.

Incrementation Designator, h (1-Bit)

The *h* designator specifies incrementation of the modifier stipulated in the *b* portion of the instruction word. When *h* equals 0, the modifier remains unchanged. When *h* equals 1, the increment portion of the index register word (Figure 3-2) is applied to the modifier portion, thereby altering the subsequent address modification.

Indirect Addressing Designator, i (1-Bit)

The *i* designator specifies either direct or indirect addressing of the operand. Indirect addressing means that the *address* of the operand rather than the operand itself is contained in the location specified by the *u* designator. Thus, *u* contains the address of an address instead of the address of an operand.

When *i* equals 0, direct addressing applies; when *i* equals 1, indirect addressing applies. In the latter case, the rightmost 22-bits contained in the location specified by the *u* designator replace the rightmost 22-bits in the current instruction. Because the *b, h, i,* and *u* designators are involved in this substitution, all indexing, incrementing, and indirect addressing operations can be cascaded.

Base Operand Address, u (16-Bits)

The *u* designator specifies the base operand address, that is one of the storage locations in memory. The base operand address *u* becomes *U* (the effective operand address) after specified indexing or indirect addressing operations have been performed.

Most instructions reference an operand in memory, except when *j* equals 16 or 17 (octal). In this case, the actual operand itself is taken directly from the *u* portion of the instruction.

The *u* designator can also be used to provide the shift count or to specify core-memory sections involved in a memory lockout.

TWO-ADDRESS ACCESSIBILITY

The index, *A,* and *R*-registers can be accessed in one of two ways. First, the film-memory address associated with any one of these registers can be accessed in the same way any film-memory address is referenced; that is, by specifying the address in an instruction's *u* designator. Second, *A, B,* and *R*-registers can be accessed via the *a* designator. In this case, the type of instruction to be executed determines which of the three groups of registers is pertinent to the operation. The value of the *a* designator specifies a particular register within a group.

Any one of 16 arithmetic registers or 16 *R*-registers may be referenced by placing the appropriate value, ranging between 0 and 15, in the *a* designator. In respect to specially assigned *R*-registers, the *a* values are as follows:

VALUE	REFERENCE
0	real-time clock
1	repeat count register
2	*M*-register
3	*T*-register

Sixteen index registers may be accessed via the *a* designator. Here again, the *a* values range between 0 and 15. However, the index register accessed via an *a* value of 0 *cannot* be referenced by the *b* designator. Accordingly, index register 0 is employed only in certain instructions; for example, *Block Transfer; Load Ba Modifier Only*; and *Test Modifier* . As previously stated, a *u* designator containing a value of 0 will inhibit address modification.

4. INTERNAL DATA PATHS

Solid-state circuitry within the Central Computer provides the data paths for all internal transfers. The specific circuits over which data moves depend primarily upon the interpretation of the various designators within the instruction word. (The role of *skip* and *jump* instructions in determining data paths is discussed in Chapters 9 and 10.)

EXECUTION CYCLE

To illustrate the functions of the various instruction word designators, assume an arithmetic instruction, stored at the address contained in *P*, is to be executed. Assume further that instructions are stored in one bank and data in the other; the *j* designator value is not equal to 16 or 17; and the *b* designator is not equal to 0. Once the arithmetic instruction has been read into *PCR,* the following events take place:

1. The *f*, *j*, and *a* designators are interpreted and the appropriate circuitry is alerted.
2. The lower half of the instruction (*h*, *j*, and *u* designators) is transferred from *PCR* to the indexing unit.
3. The *b* designator is tested to determine which index register, if any, is to participate in address modification.
4. If modification is stipulated (the contents of *b* are unequal to 0), the lower-half of the contents of the specified index register is transferred to the adder in the indexing unit.
5. The contents of the *u* designator, with two 0's placed to the immediate left, are transferred to the adder where modification takes place as 18-bit one's complement addition.
6. Concurrently, the results of the previous instruction involved in arithmetic operations are transferred from a temporary storage register within the arithmetic section to the *A*-register specified in that same instruction.
7. After modification (step 5), the two leftmost bits are dropped and the address is transferred from the adder to *SCC* where it is decoded for subsequent referencing to memory.
8. When modification is specified, the *h* designator in the current instruction is tested to determine whether the index register modifier *(Q)* is to be incremented (or decremented) by (Δ). If *h* equals 1, the increment is applied to the modifier.
9. After incrementation, the new modifier is sent into the lower half of the index register specified by the *b* designator. The increment portion remains unchanged.
10. The operand address is transferred from *SCC* (step 7) to the appropriate storage address register (S0, S1, or S2).
11. The entire 36-bit contents of the location specified in the storage address register are transferred into the appropriate memory unit's *Z*-register.
12. The *i* designator is tested to determine whether direct or indirect addressing is stipulated.
13. The contents of the *A*-register specified in the current instruction are transferred from film memory to a temporary storage register in the arithmetic section.

14. The actual data transfer, in accordance with the *j* designator interpreted in step 1, is made from memory (*Z*0, *Z*1, or *Z*2) to the arithmetic section.*

15. The program address register, *P*, is incremented by 1 to provide for the sequential execution of instructions.

16. The next instruction, stored at the address now contained in *P*, is referenced in memory.

17. The circuitry alerted by the *f* designator in step 1 performs the desired operation.

18. The next instruction, referenced in step 15, is sent to *PCR*.

19. An input-output transmission may be performed while the specified arithmetic operation (step 17) is being completed.

For most instructions, the preceding steps require 4.0 microseconds. Execution time is extended by 4.0 microseconds when the operand reference is made to the same bank as the instruction reference.

The block diagram in Figure 4-1 depicts the principal paths over which data moves during the execution cycle.

** The j designator is ineffective when the operand is read from film memory (Z0). For certain instructions, 18 bits are transferred to or from a u address specifying film memory. However, the transfer is made as specified by the h designator rather than the j.*

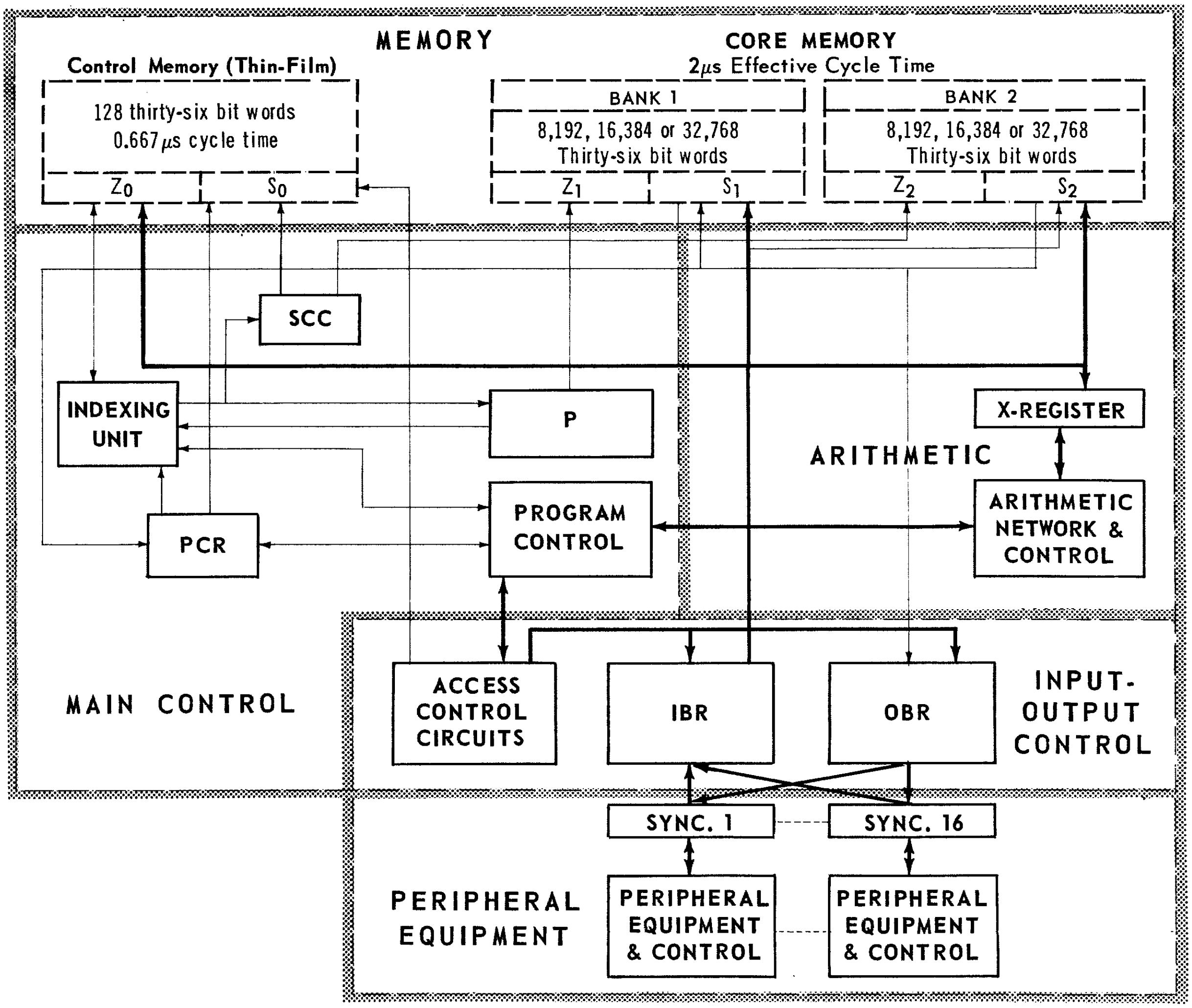


Figure 4-1. Block Diagram of the UNIVAC 1107 Thin-Film Memory Computer.

j DESIGNATOR UNEQUAL TO 16 OR 17

When the current instruction's *j* designator is neither 16 nor 17 (octal), one of eleven possible portions of a word or the entire word may serve as the operand. The *j* designator becomes effective when the operand is being transferred between core memory (*Z*1 or *Z*2) and the arithmetic section. Consequently, with respect to the *j* designator, a data transfer to or from film memory (*Z*0) will always involve a complete 36-bit word. Figure 4-2 shows the data paths utilized when the *j* designator is neither 16 nor 17.

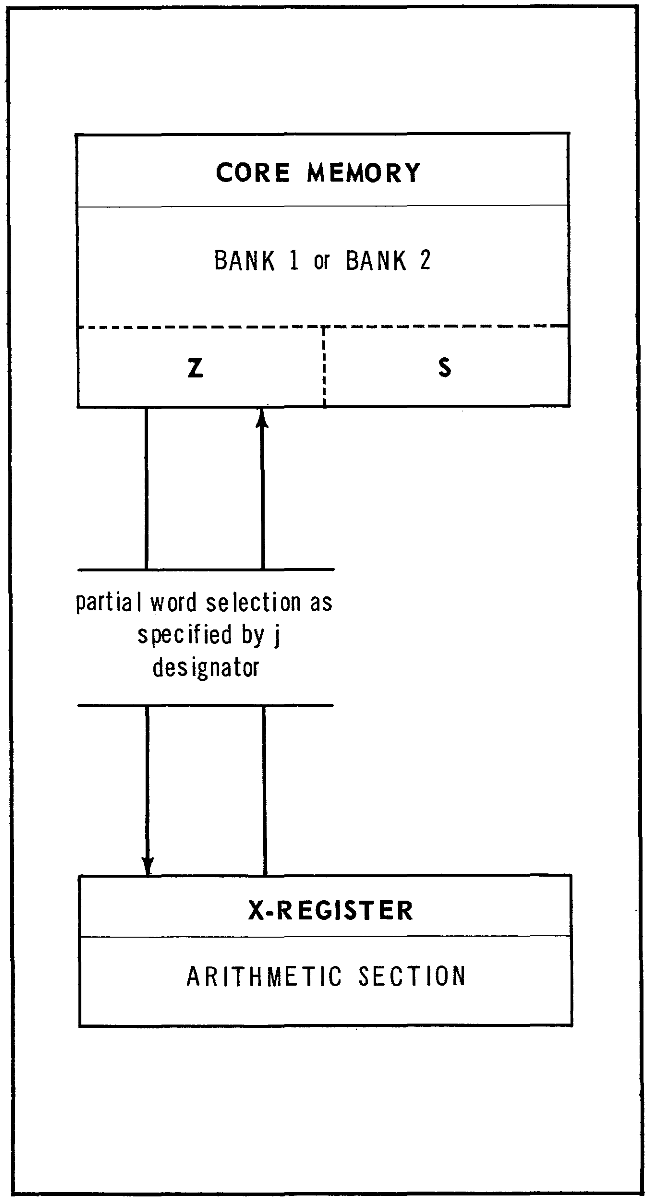


Figure 4-2. Data Paths for j Unequal to 16 or 17

j DESIGNATOR EQUAL TO 16 OR 17

The *j* designator may also stipulate that the operand is to be transferred from the instruction word itself. This operation is specified by a *j* value of 16 or 17 (octal). Then, depending upon the contents of the *b* designator, either 16 or 18 bits will be transferred from low-order positions in the current instruction to the arithmetic section.*

When *j* is 16 and *b* is not 0 (that is, index register modification is specified), the 16-bit contents of the current instruction's *u* designator serve as the operand. In the indexing unit, two binary 0's are placed to the immediate left of the 16 bits taken from the instruction. After the specified modification has been performed as 18-bit one's complement addition, the 18-bit operand is transferred to the lower half of the *X*-register for subsequent transmission to the arithmetic section. The upper half of this register is cleared to 0's.

When *j* is equal to 16 and *b* is equal to 0 (modification is inhibited), an 18-bit operand will be transferred from the instruction word to the arithmetic section. The 18 bits are taken from the current instruction's *h, i,* and *u* designators. Once again, the upper half of the *X*-register is cleared to 0's.

A *j* value of 17 is executed in a manner similar to those described, with the exception that the sign of the operand, as it enters the lower half of the *X*-register, is extended to the left. Sign extension, then, replaces the filling in of 0's.

Figure 4-3 depicts the data paths used in transferring operands from an instruction to the arithmetic section. Once the operand has entered the arithmetic section, the specified arithmetic operation is performed. Indirect addressing, if specified, is performed before transferring the operand to the arithmetic section.

INDIRECT ADDRESSING

When the current instruction's *i* designator is equal to one, an indirect addressing operation will be performed. In this case, the rightmost 22 bits contained in *Z*1, or *Z*2, are transferred (step 11 in the execution cycle) to corresponding positions in *PCR*. The execution cycle then reverts to step 2 and remains in this loop until step 13 specifies direct addressing.

** The j values of 16 and 17 are effective only in transfers to the arithmetic section. These j values inhibit transfers from the arithmetic section.*

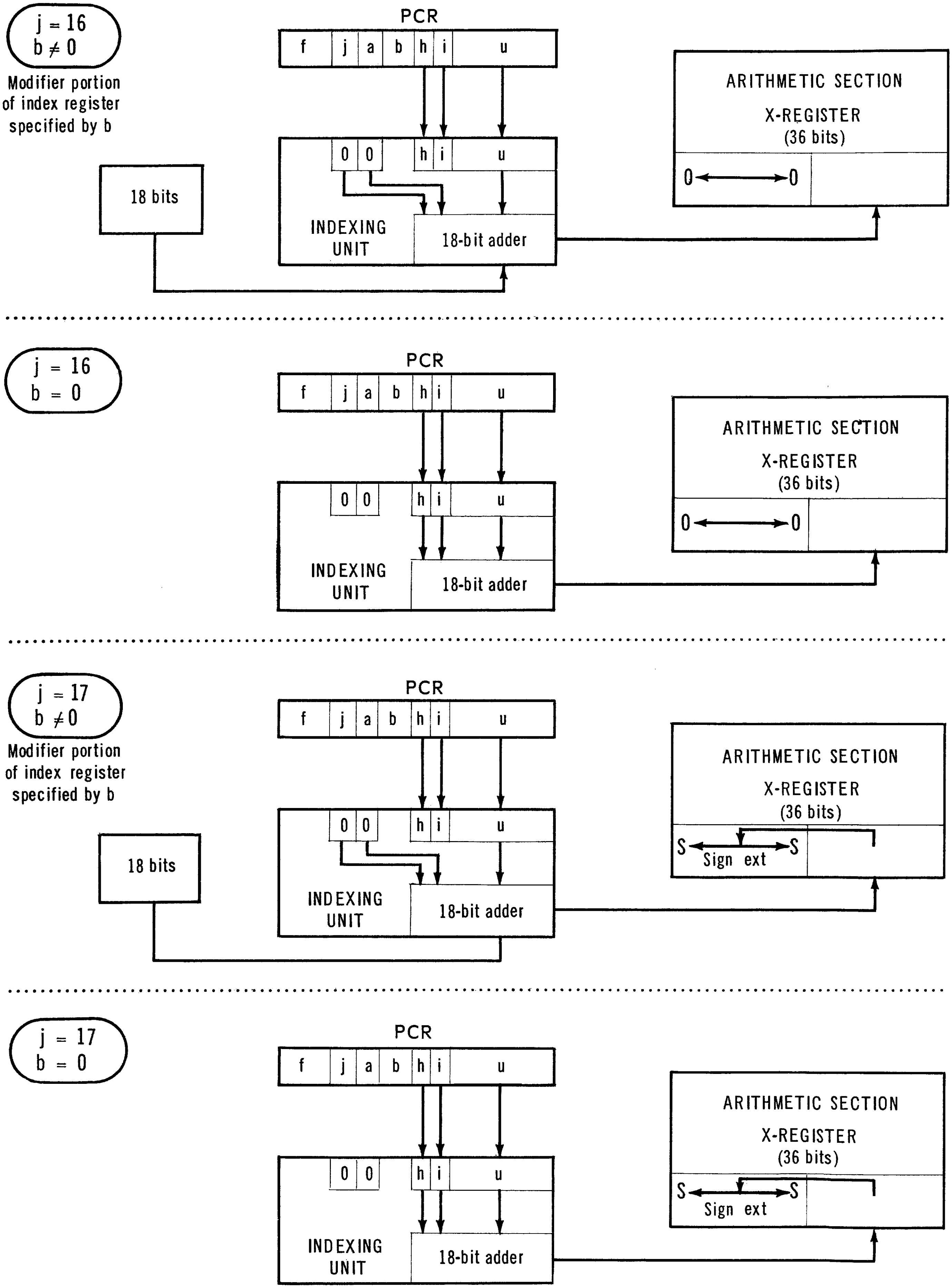


Figure 4-3. Data Paths for a j of 16 or 17.

Figure 4-4 shows the data paths utilized in indirect addressing. Note that the 22 bits cannot be transferred from film memory (*Z*0), since the *PCR* can only be entered from core storage, *Z*1 or *Z*2.

Because the 22 bits read into *PCR* include the *b*, *h*, and *i* designators, as well as a new *u* designator, indirect addressing can be cascaded.

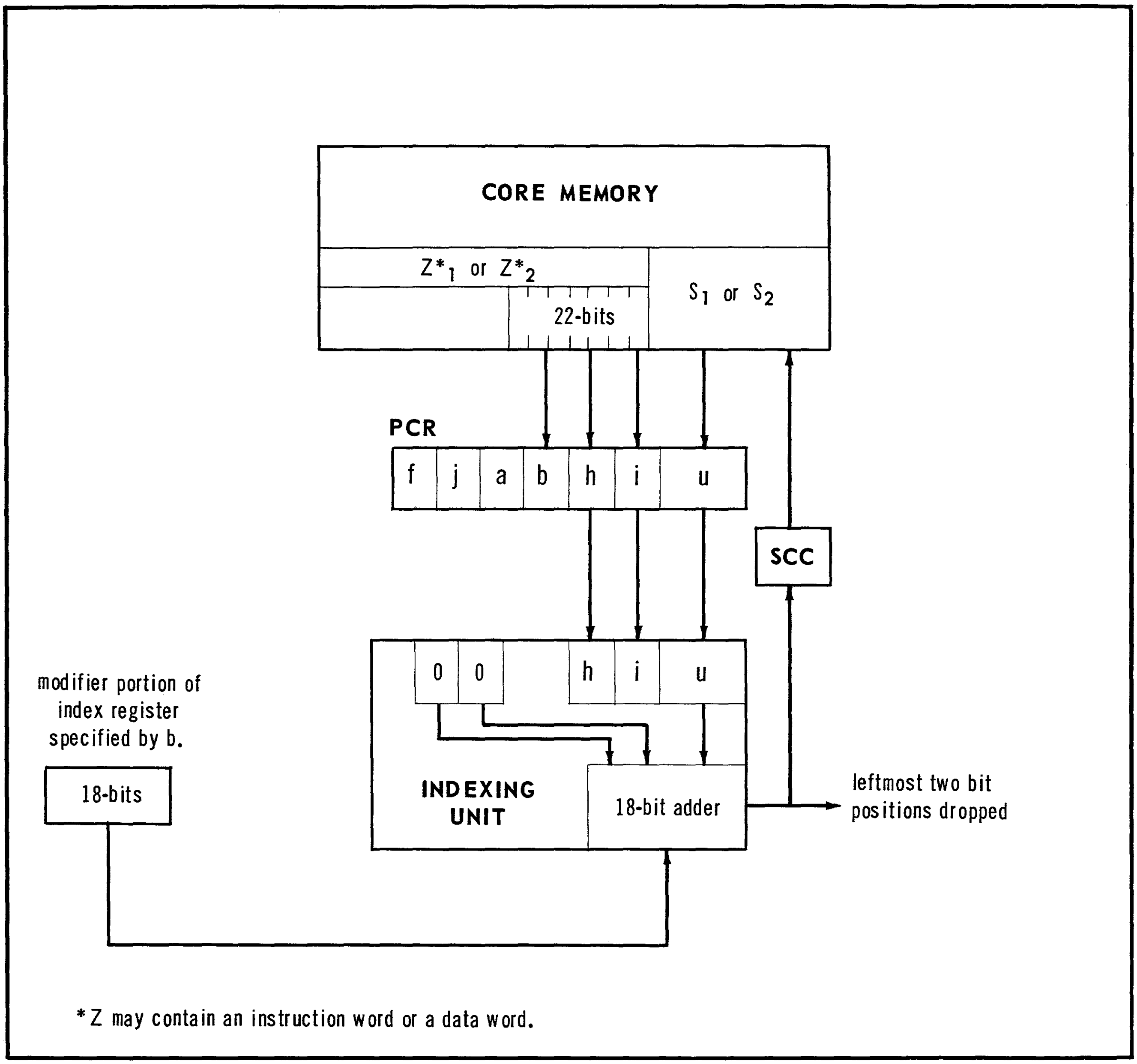

CORE MEMORY
Z*1 or Z*2
22-bits
S1 or S2
PCR
f
j
a
b
h
i
u
SCC
modifier portion of index register specified by b.
18-bits
0
0
h
i
u
INDEXING UNIT
18-bit adder
leftmost two bit positions dropped
*Z may contain an instruction word or a data word.

Figure 4-4. Data Paths for Indirect Addressing

GLOSSARY AND CONVENTIONS

Listed below are the abbreviations and symbols frequently used in the ensuing chapters.

( ) – Contents of the register or address specified within the parentheses.

( )' – The complement of the contents of the specified register or address.

|( )| – The absolute value of the contents of the specified register or address.

$(\ )_{35-00}$ – Subscript numbers indicate the range of pertinent bit positions in the word located at the specified address.

$(\ )_i$ – The initial contents of the register or address specified within the parentheses.

$(\ )_f$ – The final contents of the register or address specified within the parentheses.

u – The address specified in the current instruction's *u* designator.

U – The effective operand address.

A – One of 16 arithmetic registers as specified by the *a* designator.

A + 1 – The arithmetic register located at the address immediately following the one specified by the *a* designator.

B – One of 15 index registers as specified by the *b* designator.

B_a – One of 16 index registers as specified by the *a* designator.

R_a – One of 16 *R*-registers as specified by the *a* designator.

M – The mask register.

NI – The next sequential instruction.

→ – Transfer the word located at the address shown to the left of the arrow to the address shown on the right.

$(\ ) \odot (\ )$ – The logical product of the contents of the addresses shown to the right and left of the symbol.

$(\ ) \oplus (\ )$ – The logical sum of the contents of the addresses shown to the right and left of the symbol.

$(\ ) \overline{\oplus} (\ )$ – The logical difference between the contents of the addresses shown to the right and left of the symbol.

For the purpose of presentation, the internal instructions in the UNIVAC 1107 repertoire have been subdivided into nine groups. Within practical limits, assignment of an instruction to a particular group was contingent upon the type of operation performed by the instruction. A chapter is devoted to each of these nine groups.

In all instructions, the *b, h,* and *i* designators provide for index register modification, incrementation of the modifier, and indirect addressing. The *a* designator normally specifies one of 16 arithmetic registers (octal addresses 14 through 33). When the *a* designator specifies address 33 and a 2-word operand or result is involved, the next higher arithmetic register *(A + 1)* is address 34, an unassigned film-memory location. Note that the program does not revert back to address 14. In effect, then, the system has 17, rather than 16, arithmetic registers.

The values of *f* and *j* designators, the examples of individual instructions, and the memory addresses cited in the notes that follow many instructions are presented in the octal numbering system.

The execution time for all instructions is shown in microseconds.

Partial word transfers that do not involve sign extension (see Figures 3-7 and 3-8) are used primarily for transferring characters and not arithmetic quantities. The programmer, however, is not restricted to this usage.

The numeric representation of the operation code (*f* designator) is shown for each instruction.

When used to determine an operand the *j* designator may be expressed mnemonically as shown in the following diagram:

MNEMONICS

Sixths	S_1	S_2	S_3	S_4	S_5	S_6
Thirds	XT_1		XT_2		XT_3	
Halves	H_1			H_2		
Whole	W					

In mnemonic representation, a leading *X* signifies sign extension. In transfers to the arithmetic section, the leading *X* is mandatory with thirds and optional with halves. Sixths are never extended. In transfers from the arithmetic section, the sign bit is never extended. In this case, the leading *X* is dropped from the mnemonic representation of thirds. Similarly, it would not be used to express a half-word transfer.

5. TRANSFER INSTRUCTIONS

Thirteen instructions in the UNIVAC 1107 repertoire specify internal data transfers. These instructions may be further subdivided on the basis of whether they specify data transfers to film-memory registers (Load Instructions) or from film-memory registers (Store Instructions).

Every internal transfer utilizes the arithmetic section of the Central Computer. The entrance to the arithmetic section is made via the *X*, or exchange, register. For example, a data transfer from core memory *(Z*1 or *Z*2) to film memory (*Z*0) is accomplished automatically in the following manner:

Core *(Z*1 or *Z*2) ⟶ *X*-register ⟶ Arithmetic Section ⟶ Film (*Z*0)

Depending upon the value contained in the *j* designator, one of eleven possible portions of a word or the entire word itself will be transferred in compliance with the instruction.

In the case of a partial transfer, the *j* designator becomes effective when the 36-bit word is being transferred between core memory (*Z*1 or *Z2)* and the *X*-register. Accordingly, in transfers from film memory to core memory, the entire data word will enter the arithmetic section. However, in transfers from core memory to film memory, only the selected portion of the data word enters the arithmetic section. The partial word is then shifted to the right, while binary 0's are filled in to the left.*

LOAD

Seven instructions specify load operations. Upon execution of a load instruction, data contained in either core memory or film memory is transferred to a register in film memory.

In load instructions, the *a* designator specifies the arithmetic register, index register, or *R*-register to which data is to be transferred. The instruction itself, as evidenced by the operation code (*f* designator), determines which of the three types of registers is pertinent to the operation. The *u* designator in load instructions determines the location from which data will move.

When the *u* designator specifies film memory (*Z*0) and *j* is unequal to 16 or 17, an entire data word is transferred.

* *The sign of the selected portion of a word is extended to the left when j equals 3 through 7.*

LOAD POSITIVE

OPERATION CODE: 10

MNEMONIC CODE: LDP

OPERATION: (U) → A

DESCRIPTION: Transfer the *j*-determined portion of the contents of *U* to the specified *A*-register.

EXAMPLES:

1.
where $(A)_i$ = 3 2 1 0 4 4 4 4 0 1 2 3 and $(U)_i$ = $\boxed{\text{4 4 5 5 6 6 7 7 1 1 2 2}}$

and j = 0 THEN $(A)_f$ = $\boxed{\text{4 4 5 5 6 6 7 7 1 1 2 2}}$

2.
where $(A)_i$ = 6 6 5 0 3 2 7 1 4 3 2 5 and $(U)_i$ = $\boxed{\text{2 1 1 2 4 1 1 4 5 1 1 5}}$

and j = 0 THEN $(A)_f$ = $\boxed{\text{2 1 1 2 4 1 1 4 5 1 1 5}}$

3.
where $(A)_i$ = 5 5 5 5 4 4 4 4 3 3 3 3 and $(U)_i$ = $\boxed{\text{2 2 2 2 6 6}}$ 6 6 7 7 7 7

and j = 2 THEN $(A)_f$ = 0 0 0 0 0 0 $\boxed{\text{2 2 2 2 6 6}}$

4.
where $(A)_i$ = 3 3 3 3 2 2 2 2 1 1 1 1 and $(U)_i$ = 2 0 2 0 1 4 1 4 $\boxed{\text{5 3 3 5}}$

and j = 5 THEN $(A)_f$ = 7 7 7 7 7 7 7 7 $\boxed{\text{5 3 3 5}}$

Sign Extended

NOTES: In example 4, the *j* value causes the negative sign of a partial word to be extended in *A*.

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

LOAD NEGATIVE

OPERATION CODE: 11

MNEMONIC CODE: LDN

OPERATION: –(U) → A

DESCRIPTION: Transfer the complement of the *j*-determined portion of the contents of *U* to the specified *A*-register.

EXAMPLES:

where $(A)_i$ = 5 5 5 5 3 3 3 3 2 2 2 2 and $(U)_i$ = $\boxed{\text{1 1 1 1 0 0 0 0 6 6 6 6}}$

and j = 0 THEN $(A)_f$ = $\boxed{\text{6 6 6 6 7 7 7 7 1 1 1 1}}$

2.
where $(A)_i$ = 3 1 2 4 5 0 0 2 6 1 7 7 and $(U)_i$ = $\boxed{\text{2 5 3 1 4 3 2 7 3 5 2 4}}$

and j = 0 THEN $(A)_f$ = $\boxed{\text{5 2 4 6 3 4 5 0 4 2 5 3}}$

3.
where $(A)_i$ = 6 6 6 6 4 4 4 4 5 5 5 5 and $(U)_i$ = 1 1 1 1 6 6 $\boxed{\text{6 6 3 3 3 3}}$

and j = 3 THEN 0 0 0 0 0 0 $\boxed{\text{1 1 4 4 4 4}}$

4.
where $(A)_i$ = 1 2 3 4 3 2 1 0 1 2 3 4 and $(U)_i$ = 3 0 0 4 $\boxed{\text{5 0 0 5}}$ 6 0 0 6

and j = 6 THEN $(A)_f$ = 0 0 0 0 0 0 0 0 $\boxed{\text{2 7 7 2}}$

NOTES: Regardless of the *j* value, positive quantities in *U* will be stored negative in *A*. However, when *j* equals 0 or 3 through 7, negative quantities in *U* will be stored positive in *A*.

The *j* factor becomes effective prior to complementation in the arithmetic section.

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

LOAD POSITIVE MAGNITUDE

OPERATION CODE: 12

MNEMONIC CODE: LDM

OPERATION: $|(U)| \rightarrow A$

DESCRIPTION: Transfer the absolute value of the *j*-determined portion of the contents of *U* to the specified *A*-register.

EXAMPLES:

1.
where $(A)_i$ = 6 6 6 6 6 6 6 6 6 6 6 6
and $(U)_i$ = $\boxed{2\ 4\ 6\ 0\ 1\ 3\ 5\ 7\ 3\ 2\ 1\ 0}$
and j = 0 THEN $(A)_f$ = $\boxed{2\ 4\ 6\ 0\ 1\ 3\ 5\ 7\ 3\ 2\ 1\ 0}$

2.
where $(A)_i$ = 2 2 2 2 3 3 3 3 4 4 4 4
and $(U)_i$ = $\boxed{4\ 3\ 6\ 1\ 1\ 6\ 5\ 2\ 7\ 7\ 1\ 1}$
and j = 0 THEN $(A)_f$ = $\boxed{3\ 4\ 1\ 6\ 6\ 1\ 2\ 5\ 0\ 0\ 6\ 6}$

3.
where $(A)_i$ = 3 3 3 3 4 4 4 4 5 5 5 5
and $(U)_i$ = 5 5 5 5 2 2 2 2 $\boxed{3\ 6\ 6\ 0}$
and j = 5 THEN $(A)_f$ = 0 0 0 0 0 0 0 0 $\boxed{3\ 6\ 6\ 0}$
extend sign & complement

4.
where $(A)_i$ = 6 6 6 6 6 6 6 6 6 6 6 6
and $(U)_i$ = 3 2 0 1 $\boxed{5\ 5\ 5\ 5}$ 1 4 1 4
and j = 6 THEN $(A)_f$ = 0 0 0 0 0 0 0 0 $\boxed{2\ 2\ 2\ 2}$
extend sign & complement

NOTES: After the transfer or partial transfer of the absolute value of $(U)_i$, the 36-bit word contained in the arithmetic section is complemented if bit position 35 contains a binary 1.

EXECUTION TIMES: Alternate Bank 4.0
Same Bank 8.0

LOAD NEGATIVE MAGNITUDE

OPERATION CODE: 13

MNEMONIC CODE: LNM

OPERATION: $-|(U)| \rightarrow A$

DESCRIPTION: Transfer the complement of the absolute value of the *j*-determined portion of the contents of *U* to the specified *A*-register.

EXAMPLES:

1.
where $(A)_i$ = 3 3 3 3 4 4 4 4 5 5 5 5
and $(U)_i$ = $\boxed{6\ 6\ 6\ 6\ 1\ 1\ 1\ 1\ 2\ 2\ 2\ 2}$
and j = 0 THEN $(A)_f$ = $\boxed{6\ 6\ 6\ 6\ 1\ 1\ 1\ 1\ 2\ 2\ 2\ 2}$

2.
where $(A)_i$ = 3 3 3 3 4 4 4 4 5 5 5 5
and $(U)_i$ = $\boxed{2\ 1\ 1\ 2\ 1\ 6\ 6\ 1\ 7\ 7\ 0\ 0}$
and j = 0 THEN $(A)_f$ = $\boxed{5\ 6\ 6\ 5\ 6\ 1\ 1\ 6\ 0\ 0\ 7\ 7}$

3.
where $(A)_i$ = 3 3 3 3 4 4 4 4 5 5 5 5
and $(U)_i$ = $\boxed{1\ 1\ 1\ 1\ 2\ 2}$ 5 0 3 3 4 4
and j = 4 THEN $(A)_f$ = 7 7 7 7 7 7 $\boxed{6\ 6\ 6\ 6\ 5\ 5}$

4.
where $(A)_i$ = 3 3 3 3 4 4 4 4 5 5 5 5
and $(U)_i$ = 0 0 2 2 3 3 $\boxed{1\ 1}$ 4 4 7 7
and j = 12 THEN $(A)_f$ = 0 0 0 0 0 0 0 0 0 0 $\boxed{6\ 6}$

NOTES: When the *j*-determined portion of *U* contains a negative quantity, two complementing operations occur. The first provides the absolute value, while the second provides the complement of the absolute value. For practical purposes, note that negative quantities in the arithmetic section are transferred negative to *A*.

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

LOAD R_a

OPERATION CODE: 23

MNEMONIC CODE: LDR

OPERATION: $(U) \rightarrow R_a$

DESCRIPTION: Transfer the *j*-determined portion of the contents of *U* to the specified *R*-register.

EXAMPLES:

1.
where $(R_a)_i$ = 1 2 3 4 4 3 2 1 1 2 3 4 and $(U)_i$ = $\boxed{5\ 5\ 6\ 6\ 1\ 1\ 0\ 0\ 7\ 7\ 5\ 5}$

and j = 0 THEN $(R_a)_f$ = $\boxed{5\ 5\ 6\ 6\ 1\ 1\ 0\ 0\ 7\ 7\ 5\ 5}$

2.
where $(R_a)_i$ = 2 2 2 2 3 3 3 3 4 4 4 4 and $(U)_i$ = 4 1 1 4 5 1 1 5 $\boxed{6\ 1\ 1\ 6}$

and j = 5 THEN $(R_a)_f$ = 7 7 7 7 7 7 7 7 $\boxed{6\ 1\ 1\ 6}$

3.
where $(R_a)_i$ = 2 2 2 2 3 3 3 3 4 4 4 4 and $(U)_i$ = 4 1 1 4 5 1 1 5 $\boxed{6\ 1}$ 1 6

and j = 11 THEN $(R_a)_f$ = 0 0 0 0 0 0 0 0 0 0 $\boxed{6\ 1}$

NOTES: In this instruction, the *a* designator specifies one of sixteen *R*-registers (addresses 100 through 117).

This instruction is executed in the same manner as a Load Positive instruction.

The real-time clock may be set and the repeat count and mask registers may be loaded via this instruction.

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

LOAD B_a

OPERATION CODE: 27

MNEMONIC CODE: LDB

OPERATION: $(U) \rightarrow B_a$

DESCRIPTION: Transfer the *j*-determined portion of the contents of *U* to the specified *B*-register.

EXAMPLES:

1.
where $(B_a)_i$ = 3 3 3 3 4 4 4 4 2 2 2 2 and $(U)_i$ = $\boxed{5\ 5\ 5\ 5\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1}$

and j = 0 THEN $(B_a)_f$ = $\boxed{5\ 5\ 5\ 5\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1}$

2.
where $(B_a)_i$ = 6 5 6 5 3 1 3 1 2 7 2 7 and $(U)_i$ = $\boxed{4\ 4\ 4\ 4\ 2\ 2}$ 2 2 5 5 5 5

and j = 2 THEN $(B_a)_f$ = 0 0 0 0 0 0 $\boxed{4\ 4\ 4\ 4\ 2\ 2}$

3.
where $(B_a)_i$ = 5 4 3 2 6 5 4 3 7 6 5 4 and $(U)_i$ = 6 0 6 0 $\boxed{5\ 1\ 5\ 1}$ 4 2 4 2

and j = 6 THEN $(B_a)_f$ = 7 7 7 7 7 7 7 7 $\boxed{5\ 1\ 5\ 1}$

NOTES:

a) The *a* designator in this instruction specifies one of sixteen index registers (addresses 0 through 17).

b) This instruction is executed in a manner similar to that of the Load Positive instruction.

c) The index register into which the value is loaded by the instruction is specified by the *a* designator. With a 2-bank care system, the read *b* operation in the next instruction is (in most instructions) performed before the old "a" has been written. Therefore, the programmer should consider the possibility that the value loaded into an index register by an LDB instruction may not be available for modification purposes for the next instruction.

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

LOAD B_a MODIFIER ONLY

OPERATION CODE: 26

MNEMONIC CODE: LBM

OPERATION: $(U) \rightarrow B_{a\ 17-00}$

DESCRIPTION: Transfer the *j*-determined portion of the contents of *U* to the lower half of the specified *B*-register.

EXAMPLES:

1.
where $(B_a)_i$ = and $(U)_i$ =
300311223344 $\boxed{222244446666}$
and j = 0 THEN $(B_a)_f$ =
$300311\boxed{446666}$

2.
where $(B_a)_i$ = and $(U)_i$ =
441132566543 $0022\boxed{4400}3355$
$(Ba)_f$ =
and j = 6 THEN $441132\boxed{774400}$
Extend Sign

3.
where $(B_a)_i$ = and $(U)_i$ =
777766665555 $\boxed{3333}22224444$
$(Ba)_f$ =
and j = 7 THEN $777766\boxed{003333}$

4.
where $(B_a)_i$ = and $(U)_i$ =
454532321010 $3333114422\boxed{55}$
$(Ba)_f$ =
and j = 10 THEN $454532\boxed{000055}$

NOTES: In this instruction, the *a* designator specifies one of sixteen index registers (addresses 0 through 17). The upper half or increment portion of the specified index register always remains unchanged.

Circuitry alerted by the *f* designator (operation code 26), rather than by the *j* designator causes only the low-order 18 bits to be transferred from the arithmetic section to the specified index register. In cases where *j* designates sign extension, it should be noted that the sign is not extended beyond bit position 17. Note *c)* of the preceding instruction also applies for this instruction.

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

STORE

Six transfer instructions specify store operations. Five of these instructions entail data transfers from a film-memory register to either another film-memory location or to a core-storage location. The Store Zero instruction, on the other hand, calls for a transfer of binary 0's from the arithmetic section of the Central Computer to the specified location in film or core memory.

The *a* designator in store instructions specifies the arithmetic register, index register, or *R*-register from which data will move. The *u* designator determines the address to which the data will be transferred.

Whenever *j* equals 16 or 17 (octal) the write portion of the store operation is inhibited. In this case, operations stipulated by the instruction's *b, h,* and *i* designators will be performed. However, the actual transfer of data to the *U* address will not take place.

In the execution of a store instruction, the entire data word is first transferred from the specified arithmetic register in film memory to the arithmetic section (*X*-register). Low-order bits are then transferred from the *X*-register to those bit positions in memory (*Z*0, *Z*1, or *Z*2) specified by the *j* designator. Note that in the case of a partial transfer, the bits are always taken from the low-order positions in the *X*-register (Figure 3-8). When *U* specifies film memory (*Z*0), an entire word will be transferred.

Only the positions to which the selected bits will be transferred are affected. All other bit positions in the word located at the *U* address remain unchanged.

STORE POSITIVE

OPERATION CODE: 01

MNEMONIC CODE: STP

OPERATION: (A)→U

DESCRIPTION: Transfer the contents of the specified *A*-register to the *j*-determined positions in *U*.

EXAMPLES:

1.
where $(U)_i$ = 5 0 0 5 6 0 0 6 7 0 0 7 and $(A)_i$ = $\boxed{3\ 4\ 5\ 6\ 4\ 4\ 4\ 4\ 6\ 5\ 4\ 3}$

and j = 0 THEN $(U)_f$ = $\boxed{3\ 4\ 5\ 6\ 4\ 4\ 4\ 4\ 6\ 5\ 4\ 3}$

2.
where $(U)_i$ = 2 2 2 2 3 3 3 3 4 4 4 4 and $(A)_i$ = $\boxed{6\ 5\ 6\ 5\ 4\ 3\ 4\ 3\ 2\ 1\ 2\ 1}$

and j = 0 THEN $(U)_f$ = $\boxed{6\ 5\ 6\ 5\ 4\ 3\ 4\ 3\ 2\ 1\ 2\ 1}$

3.
where $(U)_i$ = 0 0 1 1 1 3 5 7 0 2 4 6 and $(A)_i$ = 7 7 6 6 5 5 $\boxed{4\ 4\ 3\ 3\ 2\ 2}$

and j = 2 or 4 THEN $(U)_f$ = $\boxed{4\ 4\ 3\ 3\ 2\ 2}$ 5 7 0 2 4 6

NOTES: When *j* is equal to zero a negative value in *A* will be stored negative in *U*.

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

STORE NEGATIVE

OPERATION CODE: 02

MNEMONIC CODE: STN

OPERATION: –(A)→U

DESCRIPTION: Transfer the complement of the contents of the specified *A*-register to the *j*-determined positions in *U*.

EXAMPLES:

1.
where $(U)_i$ = 2 1 2 1 3 5 3 5 7 1 7 1 and $(A)_i$ = $\boxed{1\ 2\ 3\ 4\ 5\ 6\ 7\ 0\ 1\ 2\ 3\ 4}$

and j = 0 THEN $(U)_f$ = $\boxed{6\ 5\ 4\ 3\ 2\ 1\ 0\ 7\ 6\ 5\ 4\ 3}$

2.
where $(U)_i$ = 2 1 2 1 3 5 3 5 7 1 7 1 and $(A)_i$ = $\boxed{6\ 5\ 4\ 3\ 2\ 1\ 0\ 7\ 6\ 5\ 4\ 3}$

and j = 0 THEN $(U)_f$ = $\boxed{1\ 2\ 3\ 4\ 5\ 6\ 7\ 0\ 1\ 2\ 3\ 4}$

3.
where $(U)_i$ = 0 0 0 0 1 1 1 1 2 2 2 2 and $(A)_i$ = 1 1 1 1 2 2 2 2 $\boxed{3\ 3\ 3\ 3}$

and j = 7 THEN $(U)_f$ = $\boxed{4\ 4\ 4\ 4}$ 1 1 1 1 2 2 2 2

NOTES: The entire 36-bit word contained in *A* is complemented before the *j* factor becomes effective.

When *j* equals 0, negative values in *A* will be stored positive in *U;* positive values in *A* will be stored negative in *U*.

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

STORE MAGNITUDE

OPERATION CODE: 03

MNEMONIC CODE: STM

OPERATION: $|(A)| \rightarrow U$

DESCRIPTION: Transfer the absolute value of the contents of the specified *A*-register to the *j*-determined positions in *U*.

EXAMPLES:

1.
where $(U)_i$ = 6 7 6 7 5 4 5 4 3 2 3 2 and $(A)_i$ = 3 1 3 1 2 0 2 0 4 5 4 5

and j = 0 THEN $(U)_f$ = 3 1 3 1 2 0 2 0 4 5 4 5

2.
where $(U)_i$ = 3 4 5 6 6 5 4 3 2 3 4 5 and $(A)_i$ = 5 5 0 0 3 3 0 0 2 1 2 1

and j = 0 THEN $(U)_f$ = 2 2 7 7 4 4 7 7 5 6 5 6

3.
where $(U)_i$ = 5 5 5 5 6 6 6 6 1 1 1 1 and $(A)_i$ = 4 3 2 0 7 6 5 4 1 2 3 4

and j = 6 THEN $(U)_f$ = 5 5 5 5 6 5 4 3 1 1 1 1

NOTES: The 36-bit value in *A* is complemented whenever bit position 35 contains a binary 1. Complementation occurs before the *j* value becomes effective.

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

STORE R_a

OPERATION CODE: 04

MNEMONIC CODE: STR

OPERATION: $(R_a) \rightarrow U$

DESCRIPTION: Transfer the contents of the specified *R*-register to the *j*-determined positions in *U*.

EXAMPLES:

1.
where $(U)_i$ = 2 2 2 2 6 6 6 6 7 7 7 7 and $(R_a)_i$ = 4 1 1 4 5 5 5 5 3 2 2 3

and j = 0 THEN $(U)_f$ = 4 1 1 4 5 5 5 5 3 2 2 3

2.
where $(U)_i$ = 5 7 7 5 6 0 0 6 1 1 1 1 and $(R_a)_i$ = 2 3 4 5 5 4 3 2 2 3 4 5

and j = 0 THEN $(U)_f$ = 2 3 4 5 5 4 3 2 2 3 4 5

3.
where $(U)_i$ = 2 2 2 2 6 6 6 6 7 7 7 7 and $(R_a)_i$ = 5 5 5 5 1 1 1 1 3 3 3 3

and j = 1 or 3 THEN $(U)_f$ = 2 2 2 2 6 6 1 1 3 3 3 3

NOTES: In this instruction, the *a* designator specifies one of sixteen *R*-registers (addresses 100 through 117). This instruction is executed in a manner similar to that of the Store Positive instruction.

This instruction can be used to read the real-time clock and repeat counter.

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

STORE ZERO

OPERATION CODE: 05

MNEMONIC CODE: STZ

OPERATION: $0 \rightarrow U$

DESCRIPTION: Transfer 0's to the *j*-determined portion in *U*.

EXAMPLES:

1.
where $(U)_i$ = 345665433456 and j = 0

THEN $(U)_f = \boxed{000000000000}$

2.
where $(U)_i$ = 765432111234 and j = 5

THEN $(U)_f$ = 76543211$\boxed{0000}$

NOTES: The *a* designator in this instruction is ignored. When *U* specifies film-memory and *j* is unequal to 16 or 17, the entire 36-bit word will be cleared to 0's.

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

STORE B_a

OPERATION CODE: 06

MNEMONIC CODE: STB

OPERATION: (B_a) U

DESCRIPTION: Transfer the contents of the specified *B*-register to the *j*-determined positions in *U*.

EXAMPLES:

1.
where $(U)_i$ = 204533335005 and $(B_a)_i$ = $\boxed{432412344321}$

and j = 0 THEN $(U)_f$ = $\boxed{432412344321}$

2.
where $(U)_i$ = 111133335555 and $(B_a)_i$ = 22224444$\boxed{6666}$

and j = 6 THEN $(U)_f$ = 1111$\boxed{6666}$5555

3.
where $(U)_i$ = 667755443322 and $(B_a)_i$ = 5432765432$\boxed{10}$

and j = 15 THEN $(U)_f$ = $\boxed{10}$7755443322

4.
where $(U)_i$ = 555544446666 and $(B_a)_i$ = 333300007777

and j = 16 or 17 THEN $(U)_f$ = 555544446666

NOTES: The *a* designator in this instruction stipulates one of sixteen index registers in film memory (decimal addresses 0 through 15).

When *j* equals 16 or 17, the transfer is inhibited (example 4).

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

6. ARITHMETIC INSTRUCTIONS

Eighteen instructions in the UNIVAC 1107 repertoire perform arithmetic operations. Included in this category are the basic add, subtract, multiply, and divide instructions. Special add and subtract instructions which operate in parallel upon two or three fields within a single operand are also included in this group.

ADDITION

Four arithmetic instructions specify basic add operations. Upon execution of an Add instruction, the contents of a memory location *(U)* are added to the contents of either an index register or an arithmetic register. Then, depending upon the particular type of Add instruction being executed, the result is returned to either the same index register, the same arithmetic register, or the next higher arithmetic register.

The addition (performed as a 1's complement subtractive addition) takes place in the arithmetic section of the Central Computer.

In an Add instruction, the *a* designator specifies one of sixteen arithmetic registers or one of sixteen index registers. As previously mentioned, the index register at location 0 cannot be referenced via the *b* designator.

The *j* designator in an Add instruction controls partial transfers from core memory *(Z*1 or *Z*2) to the arithmetic section. Partial transfers in conjunction with Add instructions are similar to those used with Load instructions in that whenever *U* specifies film memory and *j* is unequal to 16 or 17 (octal), an entire word will be transferred to the arithmetic section.

ADD

OPERATION CODE: 14

MNEMONIC CODE: ADD

OPERATION: $(A) + (U) \longrightarrow A$

DESCRIPTION: Add the *j*-determined portion of the contents of *U* to the contents of the specified *A*-register. Store the result in the specified *A*-register.

EXAMPLES:

1.

where $(A)_i$ = $\boxed{000003264115}$

+

and $(U)_i$ = $\boxed{000000412310}$ and j=0

THEN $(A)_f$ = 000003676425

2.

where $(A)_i$ = $\boxed{000000341221}$

+

and $(U)_i$ = 777777$\boxed{524573}$ and j=3

THEN $(A)_f$ = 000000066015

3.

where $(A)_i$ = $\boxed{000000003501}$

+

and $(U)_i$ = 77777777$\boxed{4115}$ and j=5

THEN $(A)_f$ = 777777777616

4.

where $(A)_i$ = $\boxed{345676543210}$

+

and $(U)_i$ = 0000$\boxed{3443}$1111 and j=6

THEN $(A)_f$ = 345676546653

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

ADD MAGNITUDE

OPERATION CODE: 16

FUNCTION CODE: ADM

OPERATION: $(A) + |(U)| \longrightarrow A$

DESCRIPTION: Add the absolute value of the *j*-determined portion of the contents of *U* to the contents of the specified *A*-register. Store the result in the specified *A*-register.

EXAMPLES:

1.

where $(A)_i$ = $\boxed{000023456701}$

+

and $(U)_i$ = $\boxed{000015512442}$ and j=0

THEN $(A)_f$ = 000041171343

2.

where $(A)_i$ = $\boxed{000000335555}$

+

and $(U)_i$ = $\boxed{777777444444}$ and j=0

THEN $(A)_f$ = 000000671110

3.

where $(A)_i$ = $\boxed{000025243333}$

+

and $(U)_i$ = 000000$\boxed{164507}$ and j=3

THEN $(A)_f$ = 000025430042

4.

where $(A)_i$ = $\boxed{344351156776}$

+

and $(U)_i$ = 1122$\boxed{6453}$4444 and j=6

THEN $(A)_f$ = 344351160322

NOTES: The partial word is formed in the arithmetic section before bit position 35 is tested. Then, if position 35 contains a binary 1, the partial word with sign extended is complemented to produce the absolute value.

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

ADD AND LOAD

OPERATION CODE: 20

MNEMONIC CODE: ADL

OPERATION: $(A) + (U) \longrightarrow A + 1$

DESCRIPTION: Add the *j*-determined portion of the contents of *U* to the contents of the specified *A*-register. Store the result in the next higher *A*-register.

EXAMPLES:

1.
where $(A)_i = \boxed{0\,0\,0\,0\,2\,6\,6\,5\,3\,2\,1\,1}$

$+$

and $(U)_i = 0\,0\,0\,0\,0\,0\boxed{3\,2\,7\,6\,5\,4}$ and j=3

THEN $(A+1)_f = 0\,0\,0\,0\,2\,7\,2\,0\,3\,0\,6\,5$

2.
where $(A)_i = \boxed{0\,0\,0\,0\,0\,3\,4\,5\,6\,5\,4\,3}$

$+$

and $(U)_i = 7\,7\,7\,7\,7\,7\,7\,7\boxed{5\,1\,1\,2}$ and j=5

THEN $(A+1)_f = 0\,0\,0\,0\,0\,3\,4\,5\,3\,6\,5\,6$

3.
where $(A)_i = \boxed{0\,0\,0\,0\,3\,4\,5\,6\,6\,5\,4\,3}$

$+$

and $(U)_i = 2\,4\,6\,6\boxed{5\,3\,1\,0}1\,3\,5\,7$ and j=6

THEN $(A+1)_i = 0\,0\,0\,0\,3\,4\,5\,6\,4\,0\,5\,4$

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

ADD TO B_a

OPERATION CODE: 24

MNEMONIC CODE: ADB

OPERATION: $(B_a) + (U) \longrightarrow B_a$

DESCRIPTION: Add the *j*-determined portion of the contents of *U* to the contents of the specified *B*-register. Store the result in the specified *B*-register.

EXAMPLES:

1.
where $(B_a)_i = \boxed{0\,0\,0\,0\,0\,1\,0\,3\,4\,4\,4\,4}$

$+$

and $(U)_i = \boxed{0\,0\,0\,0\,0\,1\,0\,4\,5\,5\,5\,5}$ and j=0

THEN $(B_a)_f = 0\,0\,0\,0\,0\,2\,1\,0\,2\,2\,2\,1$

2.
where $(B_a)_i = \boxed{0\,0\,0\,0\,0\,5\,0\,2\,1\,3\,5\,7}$

$+$

and $(U)_i = 7\,7\,7\,7\,7\,7\boxed{7\,7\,2\,1\,1\,3}$ and j=1

THEN $(B_a)_i = 0\,0\,0\,0\,0\,6\,0\,1\,3\,4\,7\,2$

3.
where $(B_a)_i = \boxed{0\,0\,0\,0\,1\,0\,0\,5\,4\,3\,3\,2}$

$+$

and $(U)_i = \boxed{0\,0\,0\,1\,0\,1\,0\,0\,0\,0\,0\,0}$ and j=0

THEN $(B_a)_f = 0\,0\,0\,1\,1\,1\,0\,5\,4\,3\,3\,2$

NOTES: The *a* designator in this instruction specifies one of sixteen index registers (addresses 0 through 17.)

Because the 36-bit index register word contains two distinct values (the modifier and the increment), the programmer must be certain a carry is not inadvertently made into the sign positions of the *Q* and Δ portions.

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

SUBTRACTION

Four arithmetic instructions specify subtraction. Upon execution, the contents of a memory location *(U)* are subtracted from the contents of either an index register or an arithmetic register. As determined by the operation code, the result will be stored in the same index register, the same arithmetic register, or the next higher arithmetic register.

The designators in Subtract instructions are used in the same manner as those in the Add instructions.

SUBTRACT

OPERATION CODE: 15

MNEMONIC CODE: SUB

OPERATION: (A)→(U)→A

DESCRIPTION: Subtract the *j*-determined portion of the contents of *U* from the contents of the specified *A*-register. Store the result in the specified *A*-register.

EXAMPLES:

1.
where $(A)_i$ = $\boxed{000000341233}$

−

and $(U)_i$ = $\boxed{000000156444}$ and j=0

THEN $(A)_f$ = 000000162567

2.
where $(A)_i$ = $\boxed{000000435555}$

−

and $(U)_i$ = $\boxed{777777773333}$ and j=0

THEN $(A)_f$ = 000000442221

3.
where $(A)_i$ = $\boxed{000000442222}$

−

and $(U)_i$ = $\boxed{333355}$556666 and j=4

THEN $(A)_f$ = 000000106645

4.
where $(A)_i$ = $\boxed{000000334567}$

−

and $(U)_i$ = 7777$\boxed{6666}$5555 and j=6

THEN $(A)_f$ = 000000335700

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

SUBTRACT MAGNITUDE

OPERATION CODE: 17

MNEMONIC CODE: SBM

OPERATION: (A)→|(U)|→A

DESCRIPTION: Subtract the absolute value of the *j*-determined portion of the contents of *U* from the contents of the specified *A*-register. Store the result in the specified *A*-register.

EXAMPLES:

1.
where $(A)_i$ = $\boxed{000000554444}$

−

and $(U)_i$ = $\boxed{000000036666}$ and j=0

THEN $(A)_f$ = 000000515556

2.
where $(A)_i$ = $\boxed{000044445555}$

−

and $(U)_i$ = 777773$\boxed{336666}$ and j=3

THEN $(A)_f$ = 000044106667

3.
where $(A)_i$ = $\boxed{000066662222}$

−

and $(U)_i$ = 00000000$\boxed{7444}$ and j=5

THEN $(A)_f$ = 000066661667

NOTES: The partial word is formed in the arithmetic section before bit position 35 is tested. Then, if position 35 contains a binary 1, the partial word with sign extended is complemented to produce the absolute value.

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

SUBTRACT AND LOAD

OPERATION CODE: 21

MNEMONIC CODE: SBL

OPERATION: (A) − (U)→A + 1

DESCRIPTION: Subtract the *j*-determined portion of the contents of *U* from the contents of the specified *A*-register. Store the result in the next higher *A*-register.

EXAMPLES:

1.

where $(A)_i$ = $\boxed{000000332424}$

$-$

and $(U)_i$ = $\boxed{000000145670}$ and j = 0

THEN $(A+1)_f$ = 000000164534

2.

where $(A)_i$ = $\boxed{777777754646}$

$-$

and $(A)_i$ = $77777777\boxed{2112}$ and j = 5

THEN $(A+1)_f$ = 777777752534

3.

where $(A)_i$ = $\boxed{000045543223}$

$-$

and $(U)_i$ = $7777\boxed{7666}4444$ and j = 6

THEN $(A+1)_f$ = 000045543334

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

SUBTRACT FROM B_a

OPERATION CODE: 25

MNEMONIC CODE: SBB

OPERATION: $(B_a) - (U) \rightarrow B_a$

DESCRIPTION: Subtract the *j*-determined portion of the contents of *U* from the specified *B*-register. Store the result in the specified *B*-register.

EXAMPLES:

1.

where $(B_a)_i$ = $\boxed{000004034651}$

$-$

and $(U)_i$ = $\boxed{000000002116}$ and j = 0

THEN $(B_a)_f$ = 000004032533

2.

where $(B_a)_i$ = $\boxed{000031020065}$

$-$

and $(U)_i$ = $\boxed{000012000000}$ and j = 0

THEN $(Ba)_f$ = 000017020065

NOTES: The *a* designator in this instruction is used to specify one of sixteen index registers (addresses 0 through 17).

In using this instruction, the programmer should exercise care that a carry or borrow is not inadvertently made from the increment portion (Δ) of the index register word to the modifier portion *(Q)*.

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

MULTIPLICATION

Three instructions perform multiplication. In these instructions, the *a* designator always specifies an arithmetic register. The remaining designators serve the same purpose as those used in Add instructions.

Multiplication produces a 72-bit (2-word) result. The most significant word is stored in the specified arithmetic register, while the least significant word is stored in the next higher arithmetic register. Bit position 35 in the specified *A*-register contains the sign of the 72-bit result. All 36 bit positions in *A + 1* contain data.

MULTIPLY INTEGER

OPERATION CODE: 30

MNEMONIC CODE: MPI

OPERATION: $(A) \cdot (U) \rightarrow A, A+1$

DESCRIPTION: Multiply the contents of the specified *A*-register by the *j*-determined portion of the contents of *U*. Store the most significant half of the 72-bit result in the specified *A*-register and the least significant half in the next higher *A*-register.

EXAMPLES:

1.

where $(A)_i$ = $\boxed{000000000122}$

and $(U)_i$ = $\boxed{000000000002}$ and j=0

THEN $(A)_f$ = 000000000000

AND $(A+1)_f$ = 000000000244

2.

where $(A)_i$ = $\boxed{312000000000}$

and $(U)_i$ = $002000\boxed{004444}$ and j=4

THEN $(A)_f$ = 000000000624

AND $(A+1)_f$ = 000000000000

3.

where $(A)_i$ = $\boxed{214000000000}$

and $(U)_i$ = $7777\boxed{3000}5555$ and j=6

THEN $(A)_f$ = 000000000644

AND $(A+1)_f$ = 000000000000

4.

where $(A)_i$ = $\boxed{312000000000}$

and $(U)_i$ = $\boxed{777777775777}$ and j=0

THEN $(A)_f$ = 311777777153

AND $(A+1)_f$ = 466000000000

EXECUTION TIMES: Alternate Banks 12.0
Same Bank 16.0

MULTIPLY SINGLE (INTEGER)

OPERATION CODE: 31

MNEMONIC CODE: MPS

OPERATION: (A) · (U)→A

DESCRIPTION: Multiply the contents of the specified *A*-register by the *j*-determined portion of the contents of *U*. Store the result in the specified *A*-register.

EXAMPLES:

1.

where $(A)_i$ = $\boxed{000000000122}$

and $(U)_i$ = $\boxed{000000000002}$ and j=0

THEN $(A)_f$ = 000000000244

2.

where $(A)_i$ = $\boxed{312000000000}$

and $(U)_i$ = $\boxed{002000}004444$ and j=4

THEN $(A)_f$ = 000000000000

NOTES: Basically, this instruction specifies a Multiply Integer operation. However, the least significant half of the 72-bit result (rather than the most significant half) is stored in the specified *A*-register. The 36 most significant bits are lost, while the contents of the next higher *A*-register *(A + 1)* remain unchanged.

EXECUTION TIMES: Alternate Banks 12.0
Same Bank 16.0

MULTIPLY FRACTIONAL

OPERATION CODE: 32

MNEMONIC CODE: MPF

OPERATION: (A) · (U)→A, A + 1

DESCRIPTION: Multiply the contents of the specified *A*-register by the *j*-determined portion of the contents of *U*. Store the most significant half of the fractional result in the specified *A*-register and the least significant half in the next high *A*-register.

EXAMPLES:

1.

where $(A)_i$ = $\boxed{000000000122}$

and $(U)_i$ = $\boxed{000000000002}$ and j=0

THEN $(A)_f$ = 000000000000

AND $(A+1)_f$ = 000000000510

2.

where $(A)_i$ = $\boxed{3\ 1\ 2\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}$

and $(U)_i$ = 0 0 2 0 0 0 $\boxed{0\ 0\ 4\ 4\ 4\ 4}$ and j=4

THEN $(A)_f$ = 0 0 0 0 0 0 0 0 1 4 5 0

AND $(A+1)_f$ = 0 0 0 0 0 0 0 0 0 0 0 0

3.

where $(A)_i$ = $\boxed{2\ 1\ 4\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}$

and $(U)_i$ = 7 7 7 7 $\boxed{3\ 0\ 0\ 0}$ 5 5 5 5 and j=6

THEN $(A)_f$ = 0 0 0 0 0 0 0 0 1 5 1 0

AND $(A+1)_f$ = 0 0 0 0 0 0 0 0 0 0 0 0

4.

where $(A)_i$ = $\boxed{3\ 1\ 2\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}$

and $(U)_i$ = $\boxed{7\ 7\ 7\ 7\ 7\ 7\ 7\ 7\ 5\ 7\ 7\ 7}$ and j=0

THEN $(A)_f$ = 7 7 7 7 7 7 7 7 6 3 2 7

AND $(A+1)_f$ = 0 0 0 0 0 0 0 0 0 0 0 0

NOTES: This instruction is identical to Multiply Integer with the exception that the 72-bit result is shifted one place to the left before it is stored in the two *A*-registers.

As may be seen from a comparison of examples, the left shift in Multiply Fractional doubles the result of a Multiply Integer instruction.

EXECUTION TIMES:		
	Alternate Banks	12.0
	Same Bank	16.0

DIVISION

Three arithmetic instructions stipulate division. In these instructions, the *a* designator always specifies an arithmetic register. The remaining designators are used in the same manner as with Add instructions.

The Divide Integer and the Divide Fractional instructions are normally used when the program specifies immediate division of the double-length product of a Multiply Integer or Multiply Fractional instruction.

During the execution of a Divide, overflow will occur whenever the divisor is equal to or less than the most significant half of the dividend. This condition, in turn, causes an error interrupt.

DIVIDE INTEGER

OPERATION CODE: 34

MNEMONIC CODE: DVI

OPERATION: (A, A + 1) ÷ (U) → A, A + 1

DESCRIPTION: Divide the 72-bit combined contents of the specified *A*-register and the next higher *A*-register by the *j*-determined portion of the contents of *U*. Store the quotient in the specified *A*-register and the remainder in the next higher *A*-register.

EXAMPLES:

1.

where the Dividend $(A,A+1)_i$ =

$\boxed{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0}$

÷

and the Divisor $(U)_i$ = $\boxed{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 2\ 0\ 0\ 0}$ and j = 0

Then the Quotient $(A)_f$ = 0 0 0 0 0 0 0 0 0 0 0 4

and the Remainder $(A+1)f$ = 0 0 0 0 0 0 0 0 0 0 0 0

2.

where the Dividend $(A,A+1)_i$ =

$\boxed{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 2\ 2}$

÷

and the Divisor $(U)_i$ = 0 0 0 0 0 0 $\boxed{0\ 0\ 0\ 0\ 0\ 5}$ and j=3

then the Quotient $(A)_f$ = 0 0 0 0 0 0 0 0 0 0 0 3

and the Remainder $(A+1)f$ = 0 0 0 0 0 0 0 0 0 0 0 3

3.

where the Dividend $(A,A+1)_i$ =

$\boxed{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 2\ 2}$

÷

and the Divisor $(U)_i$ = 7 7 7 7 7 7 $\boxed{7\ 7\ 7\ 7\ 7\ 2}$ and j=3

Then the Quotient $(A)_f$ = 7 7 7 7 7 7 7 7 7 7 7 4

and the Remainder $(A+1)f$ = 0 0 0 0 0 0 0 0 0 0 0 3

NOTES: Because the double-length dividend is, in effect, a single 72-bit word, the divisor need only be greater than the most significant half, that is, greater than the contents of *A*.

The remainder has the same sign as the dividend.

EXECUTION TIMES:		
	Alternate Banks	31.3
	Same Bank	35.3

DIVIDE SINGLE AND LOAD (FRACTIONAL)

OPERATION CODE: 35

MNEMONIC CODE: DVL

OPERATION: (A) − (U)→A + 1

DESCRIPTION: Divide the contents of the specified *A*-register by the *j*-determined portion of the contents of *U*. Store the result in the next higher *A*-register.

EXAMPLES:

1.
where the Dividend $(A)_i$ =

$\boxed{0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,3\,5}$

÷

and the Divisor $(U)_i$ =

$\boxed{0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,6}$ and j = 0

THEN the Quotient $(A+1)_f$ =

0 0 0 0 0 0 0 0 0 0 0 4

2.
where the Dividend $(A)_i$ =

$\boxed{0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,3\,0}$

÷

and the Divisor $(U)_i$ =

7 7 7 7 7 7 7 7 $\boxed{7\,7\,7\,2}$ and j = 5

THEN the Quotient $(A+1)_f$ =

7 7 7 7 7 7 7 7 7 7 7 3

NOTES: This instruction is similar to the Divide Fractional (operation code 36) except that only the most significant half of the dividend is used.

A remainder is not provided for by this instruction.

EXECUTION TIMES:		
	Alternate Banks	31.3
	Same Bank	35.3

DIVIDE FRACTIONAL

OPERATION CODE: 36

MNEMONIC CODE: DVF

OPERATION: (A, A + 1) ÷ (U)→A, A + 1

DESCRIPTION: Divide the 72-bit contents of the specified *A*-register and the next higher *A*-register by the *j*-determined portion of the contents of *U*. Store the quotient in the specified *A*-register and the remainder in the next higher *A*-register.

EXAMPLES:

1.
where the Dividend $(A, A+1)_i$ =

$\boxed{0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0}$

÷

and the Divisor $(U)_i$ = $\boxed{0\,0\,0\,0\,0\,0\,0\,0\,2\,0\,0\,0}$ and j = 0

Then the Quotient $(A)_f$ = 0 0 0 0 0 0 0 0 0 0 0 2

and the Remainder $(A+1)_f$ = 0 0 0 0 0 0 0 0 0 0 0 0

2.
where the Dividdnd $(A, A+1)_i$ =

$\boxed{0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,2\,2}$

÷

and the Divisor $(U)_i$ = 0 0 0 0 0 0 $\boxed{0\,0\,0\,0\,0\,5}$ and j = 3

Then the Quotient $(A)_f$ = 0 0 0 0 0 0 0 0 0 0 0 1

and the Remainder $(A+1)_f$ = 0 0 0 0 0 0 0 0 0 0 0 4

3.
where the Dividend $(A, A+1)_i$ =

$\boxed{0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,2\,2}$

÷

and the Divisior $(U)_i$ = 7 7 7 7 7 7 $\boxed{7\,7\,7\,7\,7\,2}$ and j = 3

Then the Quotient $(A)_f$ = 7 7 7 7 7 7 7 7 7 7 7 6

and the Remainder $(A+1)_f$ 0 0 0 0 0 0 0 0 0 0 0 4

NOTES: This instruction is identical to the Divide Integer with the exception that the 72-bit dividend is effectively shifted right 1-bit place prior to division. (See the notes on Divide Integer.)

As may be seen from a comparison of examples, the right shift in Divide Fractional reduces the quotient of a Divide Integer by one half.

The right shift counteracts the left shift inherent in Multiply Fractional instructions.

EXECUTION TIMES: Alternate Banks 31.3
Same Bank 35.3

MULTIPLE ADD AND SUBTRACT

Four arithmetic instructions specify parallel addition or subtraction of two or three fields within a single operand. The operation code (*f* designator) is the same for all four instructions. However, with these instructions, the *j* designator serves as a minor operation code, rather than as a partial-word determinant. The selection of a particular instruction in this group is contingent upon the value contained in *j*.

In these instructions, the *a* designator always specifies an arithmetic register.

The addition (or subtraction) of two fields is treated as two distinct operations. Bit positions 35 and 17 contain the signs of the upper and lower fields respectively. In the actual computation, carries (or borrows) are restricted to the half in which they occur. Since carries (and borrows) are not made throughout the entire 36-bit word, these instructions will not set the carry designator or the overflow designator.

Similarly, the addition (or subtraction) of three fields is treated as three distinct operations. Bit positions 35, 23, and 11 are the sign positions. Here again, the carry and overflow designators will not be set. Consequently, in all four Multiple Add and Subtract instructions, the programmer must take into account the possibility of a carry (or borrow) inadvertently made into the sign position of a particular field.

ADD HALVES

OPERATION CODE: 72

MINOR OPERATION CODE: $j = 4$

MNEMONIC CODE: ADDH

OPERATION: $(A)_{35-18} + (U)_{35-18} \rightarrow A_{35-18}$
$(A)_{17-00} + (U)_{17-00} \rightarrow A_{17-00}$

DESCRIPTION: Add the upper-half of the contents of *U* to the upper-half of the contents of the specified *A*-register. Store the result in the upper-half of the specified *A*-register. Add the lower-half of the contents of *U* to the lower-half of the contents of the specified *A*-register. Store the result in the lower-half of the specified *A*-register。

EXAMPLES:

1.

where $(A)_i$ = $\boxed{0\,0\,1\,2\,3\,4}\boxed{0\,6\,5\,4\,3\,2}$
+ +
and $(U)_i$ = $\boxed{0\,0\,0\,5\,6\,7}\boxed{0\,1\,3\,3\,3\,3}$
THEN $(A)_f$ = 0 0 2 0 2 3 1 0 0 7 6 5

2.

where $(A)_i$ = $\boxed{0\,0\,0\,2\,2\,2}\boxed{7\,7\,7\,4\,4\,4}$
+ +
and $(U)_i$ = $\boxed{7\,7\,7\,4\,4\,4}\boxed{0\,0\,5\,5\,5\,5}$
THEN $(A)_f$ = 7 7 7 6 6 6 0 0 5 2 2 2

3.

where $(A)_i$ = $\boxed{3\,3\,3\,3\,3\,3}\boxed{1\,1\,1\,1\,1\,1}$
+ +
and $(U)_i$ = $\boxed{0\,2\,4\,3\,4\,3}\boxed{1\,2\,3\,4\,5\,6}$
THEN $(A)_f$ = 3 5 7 6 7 6 2 3 4 5 6 7

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

SUBTRACT HALVES

OPERATION CODE: 72

MINOR OPERATION CODE: $j = 5$

MNEMONIC CODE: SUBH

OPERATION: $(A)_{35-18} - (U)_{35-18} \rightarrow A_{35-18}$
$(A)_{17-00} - (U)_{17-00} \rightarrow A_{17-00}$

DESCRIPTION: Subtract the upper-half of the contents of *U* from the upper-half of the contents of the specified *A*-register. Store the result in upper-half of the specified *A*-register. Subtract the lower-half of the contents of *U* from the lower-half of the contents of the specified *A*-register. Store the result in the lower-half of the specified *A*-register.

EXAMPLES:

1.

$$\begin{array}{lcl} \text{where } (A)_i & = & \boxed{0\ 4\ 5\ 5\ 5\ 5}\boxed{0\ 3\ 4\ 4\ 4\ 4} \\ & & \qquad - \qquad\qquad - \\ \text{and } (U)_i & = & \boxed{0\ 0\ 6\ 6\ 6\ 6}\boxed{0\ 0\ 5\ 7\ 5\ 7} \\ \text{THEN}(A)_f & = & 0\ 3\ 6\ 6\ 6\ 7\ 0\ 2\ 6\ 4\ 6\ 5 \end{array}$$

2.

$$\begin{array}{lcl} \text{where } (A)_i & = & \boxed{0\ 3\ 2\ 2\ 2\ 2}\boxed{7\ 7\ 6\ 2\ 2\ 2} \\ & & \qquad - \qquad\qquad - \\ \text{and } (U)_i & = & \boxed{7\ 7\ 5\ 3\ 3\ 3}\boxed{0\ 0\ 0\ 6\ 6\ 6} \\ \text{THEN}(A)_f & = & 0\ 3\ 4\ 6\ 6\ 6\ 7\ 7\ 5\ 3\ 3\ 4 \end{array}$$

3.

$$\begin{array}{lcl} \text{where } (A)_i & = & \boxed{0\ 0\ 0\ 4\ 4\ 2}\boxed{7\ 7\ 7\ 6\ 4\ 6} \\ & & \qquad - \qquad\qquad - \\ \text{and } (U)_i & = & \boxed{7\ 7\ 7\ 2\ 2\ 2}\boxed{0\ 0\ 4\ 5\ 5\ 5} \\ \text{THEN}(A)_f & = & 0\ 0\ 1\ 2\ 1\ 7\ 7\ 7\ 3\ 0\ 7\ 1 \end{array}$$

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

ADD THIRDS

OPERATION CODE: 72

MINOR OPERATION CODE: $j = 6$

MNEMONIC CODE: ADDT

OPERATION:

$$(A)_{35-24} + (U)_{35-24} \rightarrow A_{35-24}$$

$$(A)_{23-12} + (U)_{23-12} \rightarrow A_{23-12}$$

$$(A)_{11-00} + (U)_{11-00} \rightarrow A_{11-00}$$

DESCRIPTION: Add the upper third of the contents of *U* to the upper third of the contents of the specified *A*-register and store the result in the upper third of the specified *A*-register. Add the middle third of the contents of *U* to the middle third of the contents of the specified *A*-register and store the result in the middle third of the specified *A*-register. Add the lower third of the contents of *U* to the lower third of the contents of the specified *A*-register and store the result in the lower third of the specified *A*-register.

EXAMPLES:

1.

$$\begin{array}{lcl} \text{where } (A)_i & = & \boxed{0\ 4\ 4\ 4}\boxed{0\ 2\ 2\ 2}\boxed{0\ 6\ 6\ 6} \\ & & \quad + \qquad + \qquad + \\ \text{and } (U)_i & = & \boxed{0\ 2\ 3\ 4}\boxed{0\ 4\ 5\ 6}\boxed{0\ 0\ 1\ 2} \\ \text{THEN}(A)_f & = & 0\ 7\ 0\ 0\ 0\ 7\ 0\ 0\ 0\ 7\ 0\ 0 \end{array}$$

2.

$$\begin{array}{lcl} \text{where } (A)_i & = & \boxed{0\ 2\ 3\ 4}\boxed{0\ 3\ 4\ 5}\boxed{0\ 5\ 6\ 7} \\ & & \quad + \qquad + \qquad + \\ \text{and } (U)_i & = & \boxed{0\ 4\ 3\ 2}\boxed{0\ 2\ 2\ 2}\boxed{0\ 1\ 1\ 2} \\ \text{THEN}(A)_f & = & 0\ 6\ 6\ 6\ 0\ 5\ 6\ 7\ 0\ 7\ 0\ 1 \end{array}$$

3.

$$\begin{array}{lcl} \text{where } (A)_i & = & \boxed{0\ 5\ 5\ 5}\boxed{7\ 4\ 4\ 4}\boxed{0\ 3\ 3\ 3} \\ & & \quad + \qquad + \qquad + \\ \text{and } (U)_i & = & \boxed{7\ 3\ 3\ 3}\boxed{0\ 1\ 1\ 1}\boxed{7\ 4\ 4\ 4} \\ \text{THEN}(A)_f & = & 0\ 1\ 1\ 1\ 7\ 5\ 5\ 5\ 0\ 0\ 0\ 0 \end{array}$$

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

SUBTRACT THIRDS

OPERATION CODE: 72

MINOR OPERATION CODE: $j = 7$

MNEMONIC CODE: SUBT

OPERATION:

$$(A)_{35-24} - (U)_{35-24} \rightarrow A_{35-24}$$

$$(A)_{23-12} - (U)_{23-12} \rightarrow A_{23-12}$$

$$(A)_{11-00} - (U)_{11-00} \rightarrow A_{11-00}$$

DESCRIPTION: Subtract the upper thiid of the contents of *U* from the upper third of the contents of the specified *A*-register and store the result in the upper third of the specified *A*-register. Subtract the middle third of the contents of *U* from the middle third of the contents of the specified *A*-register and store the result in the middle third of the specified *A*-register. Subtract the lower third of the contents of *U* from the lower third of the specified *A*-register and store the result in the lower third of the specified *A*-register.

EXAMPLES:

1.

$$\begin{array}{lcccc} \text{where } (A)_i & = & \boxed{0344} & \boxed{0245} & \boxed{0261} \\ & & - & - & - \\ \text{and } (U)_i & = & \boxed{0056} & \boxed{0136} & \boxed{0077} \end{array}$$

THEN $(A)_f$ = 026601070162

2.

$$\begin{array}{lcccc} \text{where } (A)_i & = & \boxed{0456} & \boxed{7745} & \boxed{0034} \\ & & - & - & - \\ \text{and } (U)_i & = & \boxed{7734} & \boxed{7543} & \boxed{0045} \end{array}$$

THEN $(A)_f$ = 052102027766

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

7. LOGICAL INSTRUCTIONS

Six instructions in the UNIVAC 1107 repertoire are classified as logical instructions. Basically, these instructions entail the addition, subtraction, or multiplication of specified bit configurations rather than quantities. Logical instructions differ from arithmetic instructions in that they perform these operations in a non-arithmetic manner. In executing a logical instruction – whether it be addition, subtraction, or multiplication – the desired operation is performed on a bit-by-bit basis. The special circuitry that provides for carries from one bit position to another is not activated as in the execution of arithmetic instructions.* Consequently, carries do not occur in logical operations.

In logical instructions, the *a* designator always specifies an arithmetic register. When the result is to be stored in *A + 1* and the instruction's *a* designator specifies address 33 (the sixteenth *A*-register), the result will be stored at address 34 (an unassigned film-memory location).The program will not revert back and store the result at address 14 (the first *A*-register).

The *j* designator in logical instructions controls partial transfers from core memory (*Z*1 or *Z*2) to the arithmetic section. When *U* specifies film memory (*Z*0) and *j* is unequal to 16 or 17 (octal), an entire word will be transferred to the arithmetic section.

** The manipulation of bits on an individual basis conforms to the logic of basic computer circuitry. Consequently, instructions that stipulate the handling of data on a bit-by-bit basis are termed logical instructions.*

SELECTIVE SET

OPERATION CODE: 40

MNEMONIC CODE: SSE

OPERATION: $(A) \oplus (U) \rightarrow A + 1$

DESCRIPTION: Form the logical sum of the contents of the specified *A*-register and the *j*-determined portion of the contents of *U*. Store the result in the next higher *A*-register.

EXAMPLES:

1.

where $(A)_i = \boxed{0\,0\,0\,0\,0\,1\,4\,5\,3\,0\,3\,0}$

$\oplus$

and $(U)_i = \boxed{0\,0\,0\,0\,0\,0\,0\,0\,2\,2\,2\,2}$ and j=0

THEN $(A+1)_f$ = 0 0 0 0 0 1 4 5 3 2 3 2

2.

where $(A)_i = \boxed{0\,0\,0\,0\,0\,0\,0\,5\,5\,5\,5\,5}$

$\oplus$

and $(U)_i = 0\,0\,0\,0\,0\,0\boxed{0\,2\,3\,2\,3\,2}$ and j=3

THEN $(A+1)_f$ = 0 0 0 0 0 0 0 7 7 7 7 7

3.

where $(A)_i = \boxed{0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,2}$

$\oplus$

and $(U)_i = 0\,0\,0\,0\boxed{0\,0\,2\,4}7\,7\,3\,0$ and j=6

THEN $(A+1)_f$ = 0 0 0 0 0 0 0 0 0 0 3 6

NOTES: For practical purposes, when corresponding bit positions in *U* and *A* contain 0's, a binary 0 is placed in the same position in *A + 1*. Under all other conditions, a binary 1 is placed in the appropriate bit position in *A + 1*.

The rules governing logical binary addition are as follows:

A	0	1	1	0
U	1	0	1	0
	1	1	1	0

This type of logical binary addition is sometimes referred to as "inclusive OR"; for example, if there are bits in corresponding positions of either the *A*-register or the *U*-register OR both, then bits will be included in the corresponding position of the sum.

This instruction is normally used when the programmer wishes to buff or superimpose individual bits onto the value contained in the specified *A*-register. Binary 1's in *U* will cause binary 1's to be buffed into corresponding positions in *A + 1*, while 0's in *U* leave corresponding *A* values unchanged in A + *1*.

Initial *U* and *A* values remain unchanged.

EXECUTION TIMES:		
	Alternate Banks	4.0
	Same Bank	8.0

SELECTIVE COMPLEMENT

OPERATION CODE: 41

MNEMONIC CODE: SCP

OPERATION: $(A) \bar{\oplus} (U) \rightarrow A + 1$

DESCRIPTION: Form the logical difference of the contents of the specified *A*-register and the *j*-determined portion of the contents of *U*. Store the result in the next higher *A*-register.

EXAMPLES:

1.

where $(A)_i = \boxed{0\,0\,0\,0\,0\,0\,3\,2\,4\,4\,4\,2}$

$\bar{\oplus}$

and $(U)_i = \boxed{0\,0\,0\,0\,0\,0\,4\,1\,1\,1\,1\,1}$ and j=0

THEN $(A+1)_f$ = 0 0 0 0 0 0 7 3 5 5 5 3

2.

where $(A)_i = \boxed{0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1}$

$\bar{\oplus}$

and $(U)_i = \boxed{0\,0\,0\,0\,1\,1}7\,7\,7\,3\,3\,3$ and j=4

THEN $(A+1)_f$ = 0 0 0 0 0 0 0 0 0 0 0 0

3.

where $(A)_i = \boxed{0\,0\,0\,0\,0\,0\,2\,2\,4\,4\,4\,4}$

$\bar{\oplus}$

and $(U)_i = \boxed{7\,6\,6\,6}7\,7\,4\,4\,0\,0\,2\,1$ and j=7

THEN $(A+1)_f$ = $\overbrace{7\,7\,7\,7\,7\,7}\,5\,5\,3\,2\,2\,2$

sign extended

NOTES: When corresponding bit positions in *U* and *A* contain the same values, a binary 0 is placed in the corresponding bit position in *A + 1*. When *U* and *A* contain different values in identical bit positions, a binary 1 is placed in the corresponding position in *A + 1*.

The rules governing logical binary subtraction are as follows:

A	0	1	1	0
U	1	0	1	0
	1	1	0	0

This type of binary subtraction is sometimes referred to as "exclusive OR"; for example, if there are bits in corresponding positions of either the *A* or the *U* registers, bits will be included in the corresponding bit positions of the result; however, if corresponding bit positions of both the *A* and the *U* register contain the same value, a bit is not included in the corresponding bit position of the result.

In effect, this instruction complements those bit positions in *A* that correspond to binary 1's in *U*.

Initial *U* and *A* values remained unchanged.

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

SELECTIVE CLEAR

OPERATION CODE: 42

MNEMONIC CODE: SCL

OPERATION: $(A) \odot (U) \rightarrow A + 1$

DESCRIPTION: Form the logical product of the specified *A*-register and the *j*-determined portion of the contents of *U*. Store the result in the next higher *A*-register.

EXAMPLES:

1.

where $(A)_i = \boxed{0\,0\,0\,0\,0\,0\,0\,0\,6\,6\,6\,6}$

$\odot$

and $(U)_i = \boxed{0\,0\,0\,0\,0\,0\,0\,0\,2\,2\,2\,2}$ and j=0

THEN $(A + 1)_f = 0\,0\,0\,0\,0\,0\,0\,0\,2\,2\,2\,2$

2.

where $(A)_i = \boxed{0\,0\,0\,0\,0\,0\,0\,4\,3\,4\,5\,4}$

$\odot$

and $(U)_i = 7\,7\,7\,3\,3\,3\,0\,0\boxed{1\,1\,1\,1}$ and j=5

THEN $(A + 1)_i = 0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0$

3.

where $(A)_i = \boxed{0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1}$

$\odot$

and $(U)_i = \boxed{0\,1\,0\,1}7\,4\,4\,5\,3\,2\,3\,2$ and j=7

THEN $(A + 1)_i = 0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1$

NOTES: When corresponding bit positions in *U* and *A* contain binary 1's, a binary 1 is placed in the identical position in *A + 1*. Under all other circumstances, a binary 0 is placed in *A + 1*.

The rules governing logical binary multiplication are as follows:

A	0	1	1	0
U	1	0	1	0
	0	0	1	0

This type of logical binary multiplication is also referred to as "logical AND"; for example, if a bit is contained in corresponding positions of both the *A* and the *U* register, a bit will be included in the corresponding position of the product.

This instruction is normally used when the programmer wishes to extract or erase certain bits from the value contained in the specified *A*-register. Each binary 1 in *U* will transfer the value in the corresponding bit position in *A* to *A + 1*. A 0 in *U* causes a 0 to be inserted in the corresponding bit position in *A + 1*.

*I*nitial *U* and *A* values remain unchanged.

EXECUTION TIMES: Alternate Bank 4.0
Same Bank 8.0

SELECTIVE SUBSTITUTE

OPERATION CODE: 43

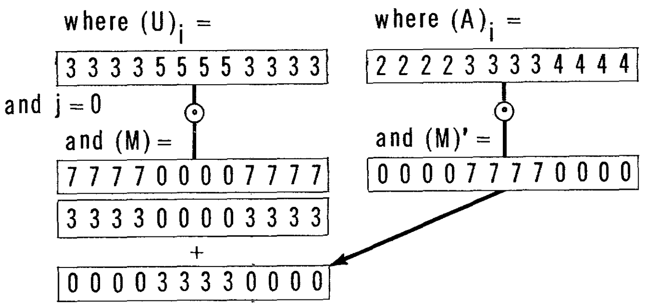
MNEMONIC CODE: SSU

OPERATION: $(U)\odot(M) + (A)\odot(M)' \rightarrow A+1$

DESCRIPTION: Form the logical product of the *j*-determined portion of *U* and the contents of the Mask register. Form the logical product of the contents of the specified *A*-register and the complement of the contents of the Mask register. Add the logical products and store the result in the next higher *A*-register.

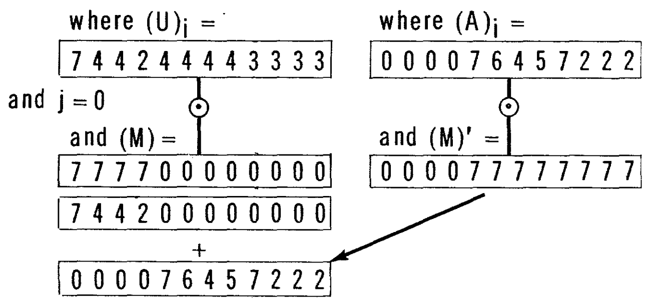
EXAMPLES:

1.



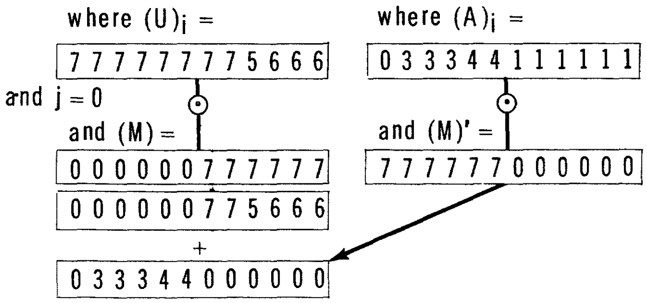
THEN $(A + 1)_f$ =
3 3 3 3 3 3 3 3 3 3 3 3

2.



THEN $(A + 1)_f$ =
7 4 4 2 7 6 4 5 7 2 2 2

3.



THEN $(A + 1)_f$ =
0 3 3 3 4 4 7 7 5 6 6 6

NOTES: Before the execution of this instruction, the desired mask is loaded into the mask register (address 102) via the Load R_a instruction.

In effect, this instruction forms a new word in *A + 1* on the basis of fields or bits selected from the words contained in *U* and *A*. Each binaty 1 in the mask will cause the value in the corresponding bit position in *U* to be set (or stored) in the same position in *A + 1*. Binary 0's in the mask will cause the values in corresponding positions in *A* to be set in identical positions in *A + 1*.

Initial *U* and *A* values remain unchanged.

EXECUTION TIMES:	Alternate Banks	4.7
	Same Bank	8.7

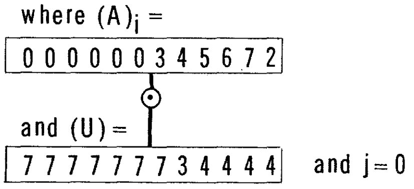
SELECTIVE EVEN PARITY TEST

OPERATION CODE:	44
MNEMONIC CODE:	SEP
OPERATION:	If (A)⊙(U) is even parity, skip NI

DESCRIPTION: If the logical product of the contents of the specified *A*-register and the *j*-determined portion of the contents of *U* is even parity, skip the next instruction. If it is odd parity, continue with the next instruction.·

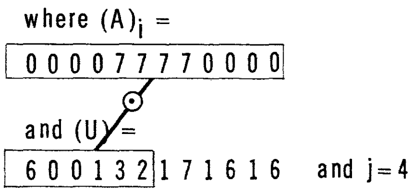
EXAMPLES:

1.



Then Parity of Logical Product
0 0 0 0 0 0 3 0 4 4 4 0 is odd (total of 5 bits)

2.



Then Parity of Logical Product

$\boxed{\underbrace{0000}_{\text{Extend sign}}77600000}$ is even (Total of 8 bits) SKIP NI

NOTES: In this instruction, *A* + *1* is not utilized.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	10.0	6.0
Same Bank	14.0	10.0

SELECTIVE ODD PARITY TEST

OPERATION CODE: 45

MNEMONIC CODE: SOP

OPERATION: If (A)⊙(U) is odd parity, skip NI

DESCRIPTION: If the logical product of the contents of the specified *A*-register and the *j*-determined portion of the contents of *U* is odd parity, skip the next instruction. If it is even parity, continue with the next instruction.

EXAMPLES:

1.

where $(A)_j$ =

$\boxed{777777554444}$

⊙

and (U) =

$\boxed{777766661111}$ and j=0

Then Parity of Logical Product

$\boxed{777766440000}$ is even (total of 18 bits)

2.

where $(A)_j$ =

$\boxed{555544445555}$

⊙

and (U) =

$\boxed{0333}66667777$ and j=7

Then Parity of Logical Product

$\boxed{000000000111}$ is odd (total of 3 bits) SKIP NI

NOTES: In this instruction, *A* + *1* is not utilized.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	10.0	6.0
Same Bank	14.0	10.0

8. SHIFT INSTRUCTIONS

Seven instructions in the UNIVAC 1107 repertoire provide for shift operations. In these instructions, the *a* designator always specifies an arithmetic register. The *j* designator in shift instructions serves as a minor operation code. The seven rightmost bits in the *u* designator provide the shift count. A shift count that exceeds 72 places will not produce a reasonable result.

Instruction execution time is independent of the number of shifts performed. In the execution of the first six shift instructions, there is only one reference to memory. Consequently, the distinction between alternate core banks and the same core bank is irrelevant.

Except for circular shifts, the least significant bits shifted out of the specified arithmetic register (or the next higher arithmetic register) are permanently lost.

SINGLE RIGHT CIRCULAR SHIFT

OPERATION CODE: 73

MINOR OPERATION CODE: $j = 0$

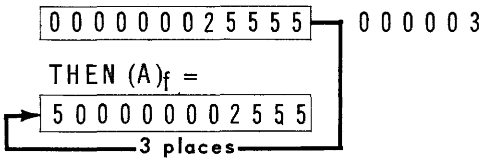
MNEMONIC CODE: SCSH

OPERATION: (A) shifted circularly *U* places to the right.

EXAMPLES:

1.

where $(A)_i$ = and U =



0 0 0 0 0 0 0 2 5 5 5 5 — 0 0 0 0 0 3

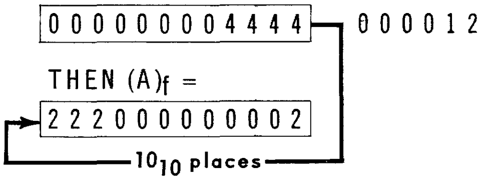
THEN $(A)_f$ =

5 0 0 0 0 0 0 0 2 5 5 5

3 places

2.

where $(A)_i$ = and U =



0 0 0 0 0 0 0 0 4 4 4 4 — 0 0 0 0 1 2

THEN $(A)_f$ =

2 2 2 0 0 0 0 0 0 0 0 2

10_{10} places

NOTES: Bits shifted out of the least significant position in *A* reappear in the most significant position and continue moving to the right until the shift is completed.

EXECUTION TIME: 4.0

DOUBLE RIGHT CIRCULAR SHIFT

OPERATION CODE: 73

MINOR OPERATION CODE: $j = 1$

MNEMONIC CODE: DCSH

OPERATION: (A, A + 1) shifted circularly *U* places to the right.

EXAMPLES:

1.
where: $(A)_i$ $(A+1)_i$ U

000000025454 | 323200000000 000030

24_{10}

THEN

000000000000 | 000254543232

$(A)_f$ $(A+1)_f$

2.
where: $(A)_i$ $(A+1)_i$ U

000000025454 | 323200000000 000074

60_{10}

THEN

000254543232 | 000000000000

$(A)_f$ $(A+1)_f$

NOTES: Bits shifted out of the least significant position in *A* enter the most significant position in *A + 1*, while bits shifted out of *A + 1* reappear in the most significant position in *A*.

EXECUTION TIME: 4.0

SINGLE RIGHT LOGICAL SHIFT

OPERATION CODE: 73

MINOR OPERATION CODE: $j = 2$

MNEMONIC CODE: SLSH

OPERATION: (A) shifted right *U* places. *U* number of 0's filled into the left.

EXAMPLES:

1.
where $(A)_i$ =

000000015673 and U = 000006

6 places →

THEN $(A)_f$ =

000000000156

2.
where $(A)_i$ =

000000022424 and U = 000005

5 places →

THEN $(A)_f$ =

000000000450

EXECUTION TIME: 4.0

DOUBLE RIGHT LOGICAL SHIFT

OPERATION CODE: 73

MINOR OPERATION CODE: *j = 3*

MNEMONIC CODE: *DLSH*

OPERATION: (A, A + 1) shifted *U* places to the right. *U* number of 0's filled in to the left.

EXAMPLES:

1.
where: $(A)_i$ $(A+1)_i$ U

000000012345 | 670000000000 000036

30_{10} places →

THEN

000000000000 | 000001234567

$(A)_f$ $(A+1)_f$

2.
where: $(A)_i$ $(A+1)_i$ U

000000012345 | 670000000000 000102

66_{10} →

THEN

000000000000 | 000000000000

$(A)_f$ $(A+1)_f$

EXECUTION TIME: 4.0

SINGLE RIGHT ARITHMETIC SHIFT

OPERATION CODE: 73

MINOR OPERATION CODE: $j = 4$

MNEMONIC CODE: SASH

OPERATION: (A) shifted right *U* places. *U* number of sign bits filled in to the left.

EXAMPLES:

1.
where $(A)_i$ =

$\boxed{000000544400}$ and U = 000006

——— 6 places ———→

THEN $(A)_f$ =

000000005444

2.
where $(A)_i$ =

$\boxed{777432100000}$ and U = 000017

——— 15_{10} places ———→

THEN $(A)_f$ =

$\boxed{777777774321}$

EXECUTION TIME: 4.0

DOUBLE RIGHT ARITHMETIC SHIFT

OPERATION CODE: 73

MINOR OPERATION CODE: $j = 5$

MNEMONIC CODE: DASH

OPERATION: (A, A + 1) shifted *U* places to the right. *U* number of sign bits filled into the left.

EXAMPLES:

1.

where: $(A)_i$	$(A + 1)_i$	U
000000121212	120000000000	000036

——— 30_{10} places ———→

000000000000	000012121212
$(A)_f$	$(A + 1)_f$

2.

where: $(A)_i$	$(A + 1)_i$	U
734522224444	000000000005	000074

——— 60_{10} places ———→

THEN

777777777777	777777777345
$(A)_f$	$(A + 1)_f$

EXECUTION TIME: 4.0

SCALE FACTOR SHIFT

OPERATION CODE: 73

MINOR OPERATION CODE: $j = 6$

MNEMONIC CODE: SFSH

OPERATION: (U)→A and shifted *left* circularly until $A_{35} \neq A_{34}$. Scaled quantity→A and shift count→A + 1.

DESCRIPTION: The contents of *U* are transferred to the specified *A*-register and shifted left circularly until bit position 35 is unequal to bit position 34. The scale-factored number is then stored in *A* and the number of shifts in *A + 1*.

EXAMPLES:

1.
where (U) =

$\boxed{000003334444}$

THEN $(A)_f$ = and $(A + 1)_f$ =

$\boxed{333444400000}$ 000000000017

15_{10} places

2.
where (U) =

$\boxed{000011447654}$

THEN $(A)_f$ = and $(A + 1)_f$ =

$\boxed{231175300000}$ 000000000015

13_{10} places

3.
where (U) =

$\boxed{777771333333}$

THEN $(A)_f$ = and $(A + 1)_f$ =

$\boxed{455555577777}$ 000000000016

14_{10} places

NOTES: This instruction will shift the contents of the specified arithmetic register a maximum of 35 places.

The number of shifts is stored in the rightmost bit positions of the next higher arithmetic register.

When bit positions 35 and 34 are initially unequal, the number is already scaled. In this case, $A + 1$ will contain 0's.

If U contains all 1's or all 0's, $(A)_f$ will equal $(U)_i$ and $(A+ 1)$ will equal 35.

Normally this instruction is used in operations involving fractional values when floating-point arithmetic is not employed.

EXECUTION TIMES:		
	Alternate Banks	6.0
	Same Bank	10.0

9. BRANCHING INSTRUCTIONS – SKIP

Forty-one instructions in the UNIVAC 1107 repertoire are classified as Branching instructions. These instructions are executed when the program reaches a point at which the selection of the next instruction depends upon certain conditions. In effect, branching instructions determine a particular program path on the basis of whether a specific condition is present. Depending upon the result of the test (or tests) inherent in the branching instruction, the program will either execute the next sequential instruction, or skip or jump to another instruction.

Twenty-three branching instructions specify skip operations.* The skip is performed in the following manner: If the branching instruction is number 21 in sequence and the test reveals the next sequential instruction is not to be executed, *P* (which contains the address of the next sequential instruction) is incremented by 1. The program then executes (or skips to) NI + 1 or instruction 23. In all cases, the program will skip only one instruction. Skip instructions are subdivided on the basis of whether or not they entail a test, a search, or a masked search.

TEST

Eleven skip instructions call for testing (or determining) the relationship between one value and another value, or between one value and two other values. Depending upon the outcome of the test, the program will skip or take the next sequential instruction.

In these instructions, the *b*, *h*, and *i* designators provide for index register modification, incrementation of the modifier, and indirect addressing. The *j* designator determines data transfers between core memory (*Z*1 or *Z*2) and the arithmetic section. The *a* designator in this instruction specifies either an arithmetic register or an index register.

* *Chapter 10 is devoted to branching instructions that stipulate a jump.*

TEST MODIFIER

OPERATION CODE: 47

MNEMONIC CODE: TMO

OPERATION: If $(B_a)_{17-00} < (U)$, take NI. If $(B_a)_{17-00} \geq (U)$, skip NI. In either case, $(B_a)_{17-00} + (B_a)_{35-18} \longrightarrow B_{a\ 17-00}$.

DESCRIPTION: If the modifier portion *(Q)* of the contents of the index register specified by the *a* designator is less than the *j*-determined portion of the contents of *U,* take the next instruction. If it is greater than or equal to the *j*-determined portion of *U,* skip the next instruction. In either case, add the increment portion (Δ) to the modifier portion *(Q)* and place the result in the modifier portion.

EXAMPLES:

1.

where U = and $(B_a)_i$ =

$\boxed{000000000006}$ 000005$\boxed{001234}$ and j=0

(Δ) (Q)

<

THEN SKIP NI AND

$(\Delta) + (Q) = (B_a)_f =$ 000005001241

2.

where U = and $(B_a)_i$ =

$\boxed{004555}$317772 000012$\boxed{004555}$ and j=4

(Δ) (Q)

≤

THEN SKIP NI AND

$(\Delta) + (Q) = (B_a)_f =$ 000012004567

3.

where (U) = and $(B_a)_i$ =

0432$\boxed{1425}$6741 777772$\boxed{000344}$ and j=6

> (Δ) (Q)

THEN TAKE NI AND

$(\Delta) + (Q) = (B_a)_f =$ 777772000337

NOTES: In this instruction, the *a* designator specifies one of sixteen index registers (addresses 0 through 17).

Subtraction is used to determine the magnitude of the appropriate fields. In this respect, a positive remainder signifies greater than, (example 1); a remainder of 0 signifies equal to (example 2); while a negative remainder signifies less than (example 3).

Only the rightmost 18 bit positions in *U* and B_a are involved in the subtraction. Consequently, when *j* equals, 0, only the lower half of *U* is pertinent to the operation.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.7	8.7
Same Bank	8.7	12.7

TEST ZERO

OPERATION CODE: 50

MNEMONIC CODE: TZR

OPERATION: If (U) = 0, skip NI

DESCRIPTION: If the *j*-determined portion of *U* is equal to 0, skip the next instruction. If it is not equal to 0, take the next instruction.

NOTES: The *a* designator is not used in this instruction.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.0	8.0
Same Bank	8.0	12.0

TEST NOT ZERO

OPERATION CODE: 51

MNEMONIC CODE: TNZ

OPERATION: If (U) = 0, skip NI

DESCRIPTION: If the *j*-determined portion of *U* is not equal to 0, skip the next instruction. If it is equal to 0, take the next instruction.

NOTES: The *a* designator is not used in this instruction.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.0	8.0
Same Bank	8.0	12.0

TEST EQUAL

OPERATION CODE: 52

MNEMONIC CODE: TEQ

OPERATION: If (U) ≠ (A), skip NI

DESCRIPTION: If the *j*-determined portion of the contents of *U* is equal to the contents of the specified *A*-register, skip the next instruction. If it is not equal, execute the next instruction.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.0	8.0
Same Bank	8.0	12.0

TEST NOT EQUAL

OPERATION CODE: 53

MNEMONIC CODE: TNE

OPERATION: If (U) ≠ (A), skip NI

DESCRIPTION: If the *j*-determined portion of the contents of *U* is not equal to the contents of the specified *A*-register, skip the next instruction. If it is equal, execute the next instruction.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.0	8.0
Same Bank	8.0	12.0

TEST LESS THAN OR EQUAL

OPERATION CODE: 54

MNEMONIC CODE: TLE

OPERATION: If (U) ≤ (A), skip NI

DESCRIPTION: If the *j*-determined portion of the contents of *U* is less than or equal to the contents of the specified *A*-register, skip the next instruction. If it is greater than the contents of *A*, execute the next instruction.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.0	8.0
Same Bank	8.0	12.0

TEST GREATER THAN

OPERATION CODE: 55

MNEMONIC CODE: TGR

OPERATION: If (U) > (A), skip NI

DESCRIPTION: If the *j*-determined portion of the contents of *U* is greater than the contents of the specified *A*-register, skip the next instruction. If it is less than or equal to the contents of *A*, execute the next instruction.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.0	8.0
Same Bank	8.0	12.0

TEST WITHIN LIMITS

OPERATION CODE: 56

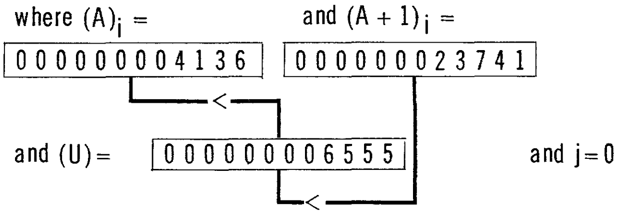
MNEMONIC CODE: TWL

OPERATION: If (A) < (U) ≤ (A + 1), skip NI

DESCRIPTION: If the *j*-determined portion of the contents of *U* is greater than the contents of the specified *A*-register but less than or equal to the contents of the next higher *A*-register, skip the next instruction. If *U* is less than or equal to *A* or greater than *A + 1*, execute the next instruction.

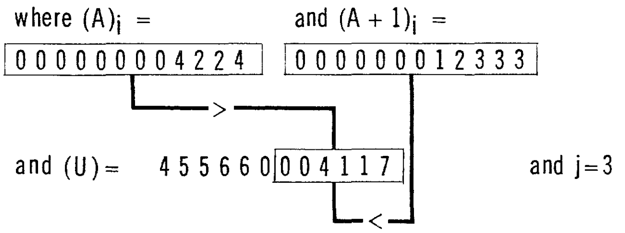
EXAMPLES:

1.



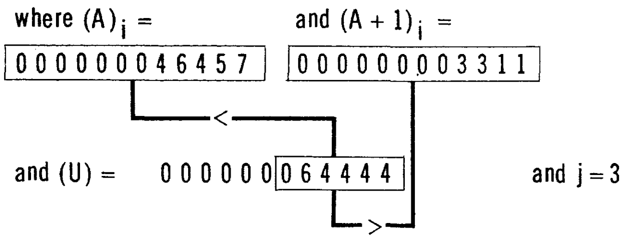
THEN SKIP NI

2.



THEN TAKE NI

3.



THEN TAKE NI

NOTES: The next instruction is skipped when the contents of the specified arithmetic register *(A)* are less than the contents of the next higher arithmetic register *(A + 1)* and the value of the contents of *U* lies between the values *(A)* and *(A + 1)*. The arithmetic registers, then serve as parameters. The contents of *A* provide the lower limit, while the contents of *A + 1*, provide the high limit.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.7	8.7
Same Bank	8.7	12.7

TEST OUTSIDE LIMITS

OPERATION CODE: 57

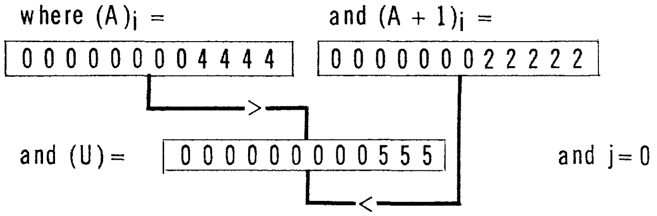
MNEMONIC CODE: TOL

OPERATION: If (U)≤(A) or (U) > (A+1), skip NI

DESCRIPTION: If the *j*-determined portion of the contents of *U* is less than or equal to the contents of the specified *A*-register or greater than the contents of the next higher *A*-register, skip the next instruction. If it is greater than the contents of *A* and less than or equal to the contents of *A + 1*, execute the next instruction.

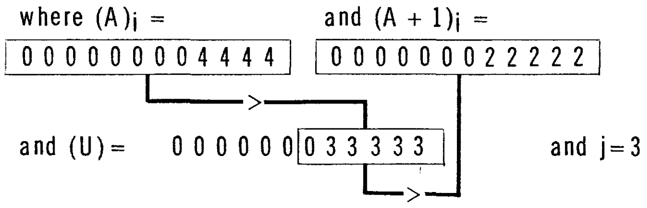
EXAMPLES:

1.



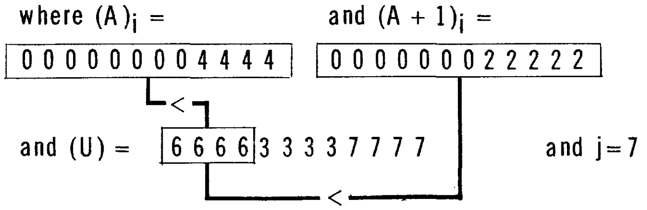
THEN SKIP NI

2.



THEN SKIP NI

3.



THEN TAKE NI

NOTES: To execute this instruction, the contents of the specified arithmetic register must be less than the contents of the next higher arithmetic register.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.7	8.7
Same Bank	8.7	12.7

TEST POSITIVE

OPERATION CODE: 60

MNEMONIC CODE: TPO

OPERATION: If (U) ≥ 0, skip NI

DESCRIPTION: If the *j*-determined portion of *U* is greater than or equal to 0, skip the next instruction. If it is less than 0, execute the next instruction.

NOTES: In this instruction, the *a* designator is not used.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.0	8.0
Same Bank	8.0	12.0

TEST NEGATIVE

OPERATION CODE: 61

MNEMONIC CODE: TNG

OPERATION: If (U) < 0, skip NI

DESCRIPTION: If the *j*-determined portion of the contents of *U* is less than 0, skip the next instruction. If it is greater than or equal to 0, take the next instruction.

NOTES: The *a* designator is not used in this instruction.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.0	8.0
Same Bank	8.0	12.0

SEARCH

Six UNIVAC 1107 instructions are classified as Search instructions. These instructions are executed in the repeat mode; that is, a particular search instruction will be executed for a specified number of times or until the object of the search is obtained, at which point the search will be terminated.

Prior to executing a search instruction, a repeat count word (Figure 3-4) is loaded into the appropriate *R*-register (address 101) via the Load R_a instruction. The *k* portion of this word contains the number of times the search instruction is to be executed.

When the main program reaches the point at which the search instruction is to be executed, the contents of *P* (the address of the next sequential instruction) are transferred automatically to address 103 (the *T*-register). Next, the *k* portion of the repeat count word is tested against 0. If it is not equal the two leftmost bit positions are dropped and the remaining 16-bit *k* is transferred to *P*. Now the actual search operation begins.

Termination Of Repeat Mode

When *k* is initially equal to 0, the contents of *PCR* (the search instruction currently being executed) are cleared to 0. The address of the next sequential instruction is then transferred from the *T*-register to *P* and the reference to memory is initiated. Accordingly, an initial *k* value of 0 prevents the execution of the search instruction.

When *k* is initially unequal to 0, the search operation is initiated in the repeat mode. Each time the search instruction is executed, *P* is reduced by 1 and tested against 0. If *P* is unequal to 0, the search instruction is re-executed. When *P* equals 0, its contents are transferred back to address 101 and the contents of the *T*-register (the address of the next sequential instruction) are returned to *P*. After referencing the *P* address to memory, the appropriate instruction is read into *PCR*, providing for the resumption of the sequential mode.

The repeat operation is terminated automatically when the object of the search is realized before *P* reaches 0. Here again, the contents of *P* are returned to the repeat count register (address 101). In this case, however, *P* contains the number of times remaining for the search instruction to be executed. The returning of *P* to address 101 provides the programmer with an opportunity for pinpointing the exact location at which the object of the search was attained. The contents of the *T*-register (the address of the next sequential instruction) are incremented by 1 – provision is thus made for the skip – and returned to *P* for referencing to memory.

Interrupts

An interrupt occuring during the execution of an instruction in the repeat mode, before the object of the search is realized, automatically causes the contents of *P* (the number of times remaining for the instruction to be executed) to be transferred back to address 101 (the repeat count register). The address of the next sequential instruction is transferred from the *T*-register to *P* and decremented by 1. After decrementation *P* contains the address of the current instruction (the search instruction). The interrupt then causes program control to jump to a fixed address for entrance into an appropriate subroutine. The first instruction in the subroutine (a Return Jump) will transfer the contents of *P* to a temporary storage location where it remains for the duration of the subroutine. The last instruction in the subroutine will stipulate a jump to the location at which *P* is stored. When the instruction stored at *P* has re-entered *PCR*, the contents of the repeat count register (the number of times remaining for the instruction to be executed) are transferred back to *P*. Provision is thus made for resuming the repeat mode at the point of interruption.

If the interrupt occurs after the object of the search has been attained but before the repeat operation has terminated, the contents of *P* are transferred back to the repeat count register. The contents of the *T*-register are incremented by 1 – thereby providing for the skip – and returned to *P*. As explained in the preceding paragraph, program control is then transferred to the appropriate subroutine. Repeat operations (*f* designators 62 – 67, and 71) require 16 microseconds, the combined setup and terminate time.

Designators

The *a* designator in a search instruction always specifies an arithmetic register. The *b, h,* and *i* designators provide for index register modification, incrementing the modifier, and indirect addressing. During the execution of the search instructions (62 – 67), the *j* designator controls data transfers between core memory (*Z*1 or *Z*2) and the arithmetic section. The *j, b, h,* and *i* designators are effective each time the search instruction is executed. It is by means of the *b* designator that the operand address is incremented (or decremented) each time the instruction is executed.

With the exception that they are executed in the repeat mode, all six search instructions (operation codes 62 through 67) are performed in the same manner as test instructions 52 through 57 respectively.

SEARCH EQUAL

OPERATION CODE: 62

MNEMONIC CODE: SEQ

OPERATION: If $(U)_i = (A)$, skip NI.
Repeat k times.

DESCRIPTION: If the *j*-determined portion of the contents of *U* is equal to the contents of the specified *A*-register, skip the next instruction. If it is not equal, re-execute the instruction until equality is sttained or *k* reaches 0. When *k* reaches 0, execute the next sequential instruction.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.0	4.0
Same Bank	4.0	4.0

SEARCH NOT EQUAL

OPERATION CODE: 63

MNEMONIC CODE: SNE

OPERATION: If $(U)_i \neq (A)$, skip NI.
Repeat k times.

DESCRIPTION: If the *j*-determined portion of the contents of *U* is unequal to the contents of the specified *A*-register, skip the next instruction. If it is equal, re-execute the instruction until inequality is attained or *k* reaches 0. When *k* reaches 0, execute the next sequential instruction.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.0	4.0
Same Bank	4.0	4.0

SEARCH LESS THAN OR EQUAL

OPERATION CODE: 64

MNEMONIC CODE: SLE

OPERATION: If $(U)_i \leq (A)$, skip NI.
Repeat k times.

DESCRIPTION: If the *j*-determined portion of the contents of *U* is less than or equal to the contents of the specified *A*-register, skip the next instruction. If it is greater than *A*, re-execute the instruction until *U* is less than or equal to *A* or *k* reaches 0. When *k* reaches 0, execute the next sequential instruction.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.0	4.0
Same Bank	4.0	4.0

SEARCH GREATER THAN

OPERATION CODE: 65

MNEMONIC CODE: SGR

OPERATION: If $(U)_i > (A)$, skip NI.
Repeat k times.

DESCRIPTION: If the *j*-determined portion of the contents of *U* is greater than the contents of the specified *A*-register, skip the next instruction. If it is less than or equal to *A*, re-execute the instruction until it is greater than *A* or *k* reaches 0. When *k* reaches 0, execute the next sequential instruction.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.0	4.0
Same Bank	4.0	4.0

SEARCH WITHIN LIMITS

OPERATION CODE: 66

MNEMONIC CODE: SWL

OPERATION: If (A) $<(U)_i \leq (A+1)$, skip NI. Repeat k times.

DESCRIPTION: If the *j*-determined portion of the contents of *U* is greater than the contents of the specified *A*-register but less than or equal to the contents of the next higher *A*-register, skip the next instruction. If *U* is less than or equal to *A* or greater than *A + 1*, re-execute the instruction until the object of the search is attained or *k* reaches 0. When *k* reaches 0, execute the next sequential instruction.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.7	4.7
Same Bank	4.7	4.7

NOTES: The contents of the specified arithmetic register should be less than the contents of the next higher arithmetic register.

SEARCH OUTSIDE LIMITS

OPERATION CODE: 67

MNEMONIC CODE: SOL

OPERATION: If $(U)_i \leq (A)$ or $(U)_i > (A+1)$, skip NI. Repeat k times.

DESCRIPTION: If the *j*-determined portion of the contents of *U* is less than or equal to the contents of the specified *A*-register or greater than the contents of the next higher *A*-register, skip the next instruction. Otherwise, re-execute the instruction until the object of the search is attained or *k* reaches 0. When k reaches 0, execute the next sequential instruction.

NOTES: The contents of the specified arithmetic register should be less than the contents of the next higher arithmetic register.

EXECUTION TIMES	NO SKIP	SKIP
Alternate Banks	4.7	4.7
Same Bank	4.7	4.7

MASKED SEARCH

Six skip instructions provide for masked search operations. Prior to execution, the mask register (address 102) is loaded with an appropriate bit configuration. Upon execution, the logical product of the mask register and the *U* address is tested against the logical product of the mask register and the *A*-register.

The masked search is similar to the search in that both operate in the repeat mode. With respect to designators, the masked search and the search are identical with the single exception that *j* in a masked search serves as a minor function code rather than as an operand determinant.

MASKED SEARCH EQUAL

OPERATION CODE: 71

MINOR OPERATION CODE: $j = 0$

MNEMONIC CODE: MSEQ

OPERATION: If $(U)_i \odot (M) = (A) \odot (M)$, skip NI. Repeat k times.

DDSCRIPTION: If the logical product of *U* and the mask register is equal to the logical product of the specified *A*-register and the mask register, skip the next instruction. If it is not equal, re-execute the instruction until equality is attained or *k* reaches 0. When *k* reaches 0, execute the next sequential instruction.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.0	4.0
Same Bank	4.0	4.0

MASKED SEARCH NOT EQUAL

OPERATION CODE: 71

MINOR OPERATION CODE: $j = 1$

MNEMONIC CODE: MSNE

OPERATION: If $(U)_i \odot (M) \neq (A) \odot (M)$, skip NI. Repeat k times.

DESCRIPTION: If the logical product of *U* and the mask register is unequal to the logical product of the specified *A*-register and the mask register, skip the next instruction. Otherwise, re-execute the instruction until inequality is attained or *k* reaches 0. When *k* reaches 0 execute the next instruction in sequence.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.0	4.0
Same Bank	4.0	4.0

MASKED SEARCH LESS THAN OR EQUAL

OPERATION CODE: 71

MINOR OPERATION CODE: $j = 2$

MNEMONIC CODE: MSLE

OPERATION: If $(U)_i \odot (M) \leq (A) \odot (M)$, skip NI. Repeat k times.

DESCRIPTION: If the logical product of *U* and the mask register is less than or equal to the logical product of the specified *A*-register and the mask register, skip the next instruction. If it is greater, re-execute the instruction until the object of the masked search is attained or *k* reaches 0. When *k* reaches 0, execute the next sequential instruction.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.0	4.0
Same Bank	4.0	4.0

MASKED SEARCH GREATER THAN

OPERATION CODE: 71

MINOR OPERATION CODE: $j = 3$

MNEMONIC CODE: MSGR

OPERATION: If $(U)_i \odot (M) > (A) \odot (M)$, skip NI. Repeat k times.

DESCRIPTION: If the logical product of *U* and the mask register is greater than the logical product of the specified *A*-register and the mask register, skip the next instruction. Otherwise, re-execute the instruction until the object of the masked search is attained or *k* reaches 0. When *k* reaches 0, execute the next sequential instruction.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.0	4.0
Same Bank	400	4.0

MASKED SEARCH WITHIN LIMITS

OPERATION CODE: 71

MINOR OPERATION CODE: $j = 4$

MENMONIC CODE: MSWL

OPERATION: If $(A) \odot (M) < (U)_i \odot (M) \leq (A+1) \odot (M)$, skip NI. Repeat k times.

DESCRIPTION: If the logical product of *U* and the mask register is greater than the logical product of the specified *A*-register and the mask register but less than or equal to the logical product of the next higher *A*-register and the mask register, skip the next instruction. Otherwise, re-execute the instruction until the object of the masked search is attained or *k* reaches 0. When *k* reaches 0, execute the next sequential instruction.

NOTES: The logical product of the specified *A*-register and the mask register should be less than the logical product of the next higher *A*-register and the mask register.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.7	4.7
Same Bank	4.7	4.7

MASKED SEARCH OUTSIDE LIMITS

OPERATION CODE: 71

MINOR OPERATION CODE: $j = 5$

MNEMONIC CODE: MSOL

OPERATION: If $(U)_i \odot (M) \leq (A) \odot (M)$ or $(U)_i \odot (M) > (A+1) \odot (M)$, skip NI. Repeat *k* times.

DESCRIPTION: If the logical product of *U* and the mask register is less than or equal to the logical product of the specified *A*-register and the mask register; or if the logical product of *U* and the mask register is greater than the logical product of the next higher *A*-register and the mask register, skip the next instruction. Otherwise, re-execute the instruction until the object of the search is attained or *k* reaches 0. When *k* reaches 0, execute the next sequential instruction.

NOTES: See notes for Masked Search Within Limits.

EXECUTION TIMES:	NO SKIP	SKIP
Alternate Banks	4.7	4.7
Same Bank	4.7	4.7

10. BRANCHING INSTRUCTIONS – JUMP

Eighteen branching instructions specify jump operations. The jump is performed in the following manner: when conditions are such that the next sequential instruction is not to be executed, the program will jump to the instruction stored at the address contained in *U*. To execute the jump, the *U* address is transferred to *P*. The contents of *U* are then transferred to *PCR* for execution. Certain instructions automatically entail the jump operation.

Unless otherwise stated, the *b*, *h*, and *i* designators provide for index register modification, incrementation of the modifier, and indirect addressing. The *j* designator normally serves as a minor operation code, while the *a* designator normally specifies an arithmetic register.

The *u* designator in jump instructions specifies the address of an instruction rather than the address of an operand. When *U* specifies an address that may be found in either film or core memory (0 – 177 octal) and a jump is called for, the next instruction will be taken from the specified address in core memory (*Z*1).

INDEX JUMP

OPERATION CODE: 70

MNEMONIC CODE: IXJP

OPERATION: If $(CM)_{ja} > 0$, jump to U. If $(CM)_{ja} \leq 0$, take NI. In either case, $(CM)_{ja} - 1 \longrightarrow CM_{ja}$.

DESCRIPTION: If the contents of the specified control-memory location are greater than 0, jump to the instruction stored at the *U* address. Otherwise, execute the next instruction. In either case, subtract 1 from the contents of the control-memory location and return the result to the specified control-memory location.

EXAMPLES:

1.

where $(CM_{ja})_i$	=	0000 0000 0055
THEN JUMP TO U		
AND $(CM_{ja})_f$	=	0000 0000 0054

2.

where $(CM_{ja})_i$	=	0000 0000 0000
THEN TAKE NI		
AND $(CM_{ja})_f$	=	7777 7777 7776

NOTES: In this instruction, the *j* and *a* designators are combined to provide the address of any one of the 128 locations in control-memory.

EXECUTION TIMES:	NO JUMP	JUMP
Alternate Banks	8.0	4.0
Same Bank	8.0	4.0

RETURN JUMP

OPERATION CODE: 72

MINOR OPERATION CODE: *j = 1*

MNEMONIC CODE: RTJP

OPERATION: (P)⟶U_{17-00}, jump to U + 1

DESCRIPTION: The contents of *P* are written into the lower-half of the word stored at *U*. The main program then jumps to the instruction stored at *U + 1*.

NOTES: In this instruction, the *a* designator is not used.

P contains the address of the next instruction.

The upper-half of the contents of *U* remains unchanged.

Because *P* contains 16 bit positions and the instruction employs a half-word (18 bits) write, bit positions 16 and 17 in *U* are forced to 0. Consequently, the *h* and *i* designators in the instruction stored at U_f inhibit incrementation of the modifier and indirect addressing.

The execution of this instruction always entails a jump operation.

In effect, the Return Jump combines, in a single instruction, the address of a subroutine exit and the transfer of control to the subroutine itself.

If *U* contains a film-memory address, a full word write of the contents of *P* preceded by 0's is made into the specified film-memory address and control is transferred to the next following address *(U + 1)* in core-memory.

EXECUTION TIMES:		
	Alternate Banks	8.0
	Same Bank	8.0

POSITIVE BIT CONTROL JUMP

OPERATION CODE: 72

MINOR OPERATION CODE: *j* = 2

MNEMONIC CODE: PBJP

OPERATION: If $(A)_{35}$ = 0, jump to U. Always shift (A) left 1 circularly.

DESCRIPTION: If bit position 35 of the word contained in the specified *A*-register is equal to 0. jump to *U*. If it is not equal, take the next instruction. In either case, the contents of *A* are shifted circularly one place to the left.

EXAMPLES:

1.

where $(A)_i$	=	0000 0002 4444
THEN JUMP TO U		
AND $(A)_f$	=	0000 0005 1110

2.

where $(A)_i$	=	7777 7733 2222
THEN TAKE NI		
AND $(A)_f$	=	7777 7666 4445

NOTES: The main program will jump to the instruction stored in the *U* address when the quantity contained in *A* is positive. When *A* contains a negative quantity, the program will execute the next sequential instruction.

EXECUTION TIMES:	NO JUMP	JUMP
Alternate Banks	4.0	8.0
Same Bank	4.0	8.0

NEGATIVE BIT CONTROL JUMP

OPERATION CODE: 72

MINOR OPERATION CODE: *j* = 3

MNEMONIC CODE: NBJP

OPERATION: If $(A)_{35}$ = 1, jump to U. Always shift (A) left 1 circularly.

DESCRIPTION: If bit position 35 of the word contained in the specified *A*-register is equal to 1, jump to *U*. If it is not equal, take the next instruction. In either case, the contents of *A* are shifted circularly one place to the left.

EXAMPLES:

1.

where $(A)_i$	=	7777 7755 3333
THEN JUMP TO U		
AND $(A)_f$	=	7777 7732 6667

2.
where $(A)_i$ = 0000 2222 4444
THEN TAKE NI
AND $(A)_f$ = 0000 4445 1110

NOTES: When the value contained in *A* is negative, the program will jump to the instruction stored at the *U* address. The next sequential instruction is executed when *A* contains a positive value.

EXECUTION TIMES:	NO JUMP	JUMP
Alternate Vanks	4.0	8.0
Same Bank	4.0	8.0

ZERO JUMP

OPERATION CODE: 74

MINOR OPERATION CODE: $j = 0$

MNEMONIC CODE: ZRJP

OPERATION: If (A) = 0, jump to U

DESCRIPTION: If the contents of the specified *A*-register are equal to 0, jump to the instruction stored at the *U* address. If not equal, take the next sequential instruction.

EXECUTION TIMES:	NO JUMP	JUMP
Alternate Banks	4.0	8.0
Same Bank	4.0	8.0

NON-ZERO JUMP

OPERATION CODE: 74

MINOR OPERATION CODE: $j = 1$

MNEMONIC CODE: NZJP

OPERATION: If (A) ≠ 0, jump to U

DESCRIPTION: If the contents of the specified *A*-register are unequal to 0, jump to the instruction stored at the *U* address. Otherwise, take the next sequential instruction.

EXECUTION TIMES:	NO JUMP	JUMP
Alternate Banks	4.0	8.0
Same Bank	4.0	8.0

POSITIVE JUMP

OPERATION CODE: 74

MINOR OPERATION CODE: $j = 2$

MNEMONIC CODE: POJP

OPERATION: If (A) ≥ 0, jump to U

DESCRIPTION: If the contents of the specified *A*-register are greater than or equal to 0, jump to the instruction stored at the *U* address. Otherwise take the next sequential instruction.

EXECUTION TIMES	NO JUMP	JUMP
Alternate Banks	4.0	8.0
Same Bank	4.0	8.0

NEGATIVE JUMP

OPERATION CODE: 74

MINOR OPERATION CODE: $j = 3$

MNEMONIC CODE: NGJP

OPERATION: If (A) < 0, jump to U

DESCRIPTION: If the contents of the specified *A*-register are less than 0, jump to the instruction stored at the *U* address. Otherwise, take the next sequential instruction.

EXECUTION TIMES:	NO JUMP	JUMP
Alternate Banks	4.0	8.0
Same Bank	4.0	8.0

CONSOLE SELECTIVE JUMP

OPERATION CODE: 74

MINOR OPERATION CODE: $j = 4$

MNEMONIC CODE: CSJP

OPERATION:If the 4-bit contents of the *a* designator are equal to the key setting on the console (1 to 15), jump to *U*. Otherwise, take the next sequential instruction.

NOTES: In this instruction, the contents of the *a* designator do not refer to an arithmetic register.

When the 4-bit contents of the *a* designator equal 0, an unconditional jump is made to the instruction stored at *U*.

EXECUTION TIMES:	NO JUMP	JUMP
	4.0	4.0

SELECTIVE STOP JUMP

OPERATION CODE: 74

MINOR OPERATION CODE: $j = 5$

MNEMONIC CODE: SSJP

OPERATION: If any of the 4 bits of the *a* designator correspond to a stop-key setting on the console (1 of 4), the Computer comes to an orderly stop. On restart, jump to the instruction stored at *U*. If the stop condition is not met, jump to the instruction stored at *U*.

Bit positions of the *a* designator and the corresponding stop keys are as follows:

WHEN: a_0 = 1 and stop key 1 is set.
a_1 = 1 and stop key 2 is set.
a_2 = 1 and stop key 3 is set.
a_3 = 1 and stop key 4 is set.

NOTES: The *a* designator in this instruction does not refer to an arithmetic register.

When the *a* designator is equal to 0, an unconditional stop is made.

EXECUTION TIMES:	NO JUMP	JUMP
	4.0	4.0

EVEN JUMP

OPERATION CODE: 74

MINOR OPERATION CODE: $j = 10$

MNEMONIC CODE: EVJP

OPERATION: If $(A)_0 = 0$, jump to U

DESCRIPTION: If bit position 0 (the rightmost bit) of the word contained in the specified *A*-register is equal to 0, jump to the instruction stored at the *U* address. If it is not equal, take the next sequential instruction.

EXECUTION TIMES:	NO JUMP	JUMP
Alternate Banks	4.0	8.0
Same Bank	4.0	8.0

ODD JUMP

OPERATION CODE: 74

MINOR OPERATION CODE: $j = 11$

MNEMONIC CODE: ODJP

OPERATION: If $(A)_0 = 1$, jump to U

DESCRIPTION: If bit position 0 (the rightmost bit) of the word contained in the specified *A*-register is equal to 1, jump to the instruction stored at the *U* address. If it is not equal, take the next sequential instruction.

EXECUTION TIMES:	NO JUMP	JUMP
Alternate Banks	4.0	8.0
Same Bank	4.0	8.0

MODIFIER JUMP

OPERATION CODE: 74

MINOR OPERATION CODE: $j = 12$

MNEMONIC CODE: MOJP

OPERATION: If $(B_a)_{17-00} > 0$, jump to U. If $(B_a)_{17-00} \leq 0$, take NI. In either case, $(B_a)_{17-00} + (B_a)_{35-18} \longrightarrow B_{a\,17-00}$.

DESCRIPTION: If the modifier portion *(Q)* of the contents of index register specified by the *a* designator is greater than 0, jump to the instruction stored at the *U* address. If it is less than or equal to 0, take the next sequential instruction. In either case, add the increment portion (Δ) to the modifier portion *(Q)* and store the result in the modifier portion.

EXAMPLES:

1. where $(B_a)_i$ = 0000 0400 3333
   THEN JUMP TO U
   AND $(B_a)_f$ = 0000 0400 3337
2. where $(B_a)_i$ = 0000 0577 7741
   THEN TAKE NI
   AND $(B_a)_f$ = 0000 0577 7746

NOTES: In this instruction, the *a* designator specifies one of sixteen index registers (addresses 0 through 17).

The increment is added to the modifier after the test has been performed.

EXECUTION TIMES:	NO JUMP	JUMP
Alternate Banks	4.0	8.0
Same Bank	4.0	8.0

LOAD MODIFIER AND JUMP

OPERATION CODE: 74

MINOR OPERATION CODE: $j = 13$

MNEMONIC CODE: LMJP

OPERATION: (P) → $B_{a\,17-00}$ and jump to U.

DESCRIPTION: The contents of *P* (the address of the next instruction) are stored in the modifier portion *(Q)* of the contents of the specified index register. The main program then jumps to the instruction stored at *U*.

NOTES: In this instruction, the *a* designator specifies one of sixteen index registers (addresses 0 through 17).

Because this instruction utilizes a half-word write, the increment portion (Δ) of the word contained in the specified index register is undisturbed.

This instruction automatically specifies a jump to the *U* address.

EXECUTION TIMES:		
	Alternate Banks	4.0
	Same Bank	4.0

OVERFLOW JUMP

OPERATION CODE: 74

MINOR OPERATION CODE: $j = 14$

MNEMONIC CODE: OVJP

OPERATION: Jump to *U* if overflow condition is set; otherwise take the next sequential instruction。

NOTES: This instruction does not utilize the *a* designator.

Overflow conditions may be set by instructions containing operation codes 14 through 21, 24, and 25. (See page 2–6 for explanation of conditions causing overflow.)

EXECUTION TIMES:	NO JUMP	JUMP
Alternate Banks	4.0	4.0
Same Bank	4.0	4.0

NO-OVERFLOW JUMP

OPERATION CODE: 74

MINOR OPERATION CODE: $j = 15$

MNEMONIC CODE: NOJP

OPERATION: Jump to *U* if overflow condition is not set; otherwise take the next sequential instruction.

NOTES: See *Overflow Jump.*

EXECUTION TIMES:	NO JUMP	JUMP
Alternate Banks	4.0	4.0
Same Bank	4.0	4.0

CARRY JUMP

OPERATION CODE: 74

MINOR OPERATION CODE: $j = 16$

MNEMONIC CODE: CYJP

OPERATION: Jump to *U* if carry condition is set; otherwise take the next sequential instruction.

NOTES: See *Overflow Jump* notes.

(See page 2–6 for an explanation of conditions which set the carry designators.)

EXECUTION TIMES:	NO JUMP	JUMP
Alternate Banks	4.0	4.0
Same Bank	4.0	4.0

NO-CARRY JUMP

OPERATION CODE: 74

MINOR OPERATION CODE: $j = 17$

MNEMONIC CODE: NCJP

OPERATION: Jump to *U* if carry condition is not set; otherwise take the next sequential instruction.

NOTES: See *Overflow Jump* notes.

EXECUTION TIMES:	NO JUMP	JUMP
Alternate Banks	4.0	4.0
Same Bank	4.0	4.0

11. BLOCK TRANSFER INSTRUCTION

The *Block Transfer* instruction is used to transfer a specified number of words from one internal memory area *(w)* to another *(v)*. Prior to execution, the appropriate repeat count word is loaded into the repeat count register (address 101) via the *Load* R_a instruction. The *k* portion of this specifies the number of words to be transmitted.

u Designator

The *u* designator in this instruction specifies the base address modified by an index register.

b Designator

The *b* designator controls the index register modification of both the *W* and the *V* addresses. When *b* contains all 0's, indexing of the *W* and *V* addresses does not occur. In this case, the a designa- is ignored.

When *b* is unequal to 0, the modifier portion of the index register specified by *b* is applied to the *U* address. The resulting address is the location in memory from which data will move (*w*).

a Designator

In this instruction, the *a* designator specifies one of sixteen index registers. When *b* is unequal to 0, the modifier portion of the index register specified by *a* is applied to the *U* address. This time, modification produces the address in memory to which data will move *(v)*.

h Designator

The *h* designator in this instruction controls the incrementing of the modifier portions of both the index register specified by *b* and the index register specified by *a*.

j Designator

The *j* designator in this instruction serves as an operand determinant. In this instruction, *j* specifies the bit positions from which data will move, as well as the bit positions to which data will move. When *j* equals 16 or 17 (octal), the write at the *V* address is inhibited. However, index registers are incremented.

Table 2 shows the *j* values and corresponding data transmissions when the block transfer is made from core memory to core memory. In this type of transfer, the non-selected portions of the words contained in the *V* addresses remain unchanged.

When the block transfer is made from core memory to film memory, 36 bits will always be contained in the film-memory location.

i Designator

In this instruction, the *i* designator is used to specify indirect addressing. Cascading of indirect addresses may be employed. However, a data transfer will not be made until *i* equals 0.

When indirect addressing is specified *(i* equals 1), the address contained in the *u* designator, modified by the index register specified by the *b* designator, provides the address from which the lower-order 22-bits will be read. When *i* equals 0, the actual data transfer is begun, using the *u*, *h*, and *b* designators most recently read into *PCR*.

Repeat Count

The repeat count is handled in the same manner as that described in the explanation of Search instructions. In executing a Block Transfer, the repeat mode will terminate only when *k* reaches 0 or an interrupt occurs. An interrupt will terminate the Block Move in a manner that will allow it to resume at the actual point of interruption (*P* decremented by 1 before transmission to temporary working storage). When *k* initially equals 0, the Block Transfer is inhibited and the next sequential instruction initiated. In this case the address of the next instruction will be taken from the lower half of film-memory location 103.

Sequence Of Events

Once the actual data transfer has begun, the sequence of events is as follows:

1. Add B_b lower to *u*.
2. Read the word *(W)* stored at the address formed in step 1.
3. Add B_b upper to B_b lower and store the result in B_b lower.

j = 0	$W_{35-00} \rightarrow V_{35-00}$	j = 10	$W_{05-00} \rightarrow V_{05-00}$
1	$W_{17-00} \rightarrow V_{17-00}$	11	$W_{11-06} \rightarrow V_{11-06}$
2	$W_{35-18} \rightarrow V_{35-18}$	12	$W_{17-12} \rightarrow V_{17-12}$
3	$W_{17-00} \rightarrow V_{17-00}$	13	$W_{23-18} \rightarrow V_{23-18}$
4	$W_{35-18} \rightarrow V_{35-18}$	14	$W_{29-24} \rightarrow V_{29-24}$
5	$W_{11-00} \rightarrow V_{11-00}$	15	$W_{35-30} \rightarrow V_{35-30}$
6	$W_{23-12} \rightarrow V_{23-12}$	16	No Transfer
7	$W_{35-24} \rightarrow V_{35-24}$	17	No Transfer

Table 2. Partial Word Designator in Block Transfer

4. Add B_a lower to u.

5. Write the word read in step 2 into memory at the address formed in step 4.

6. Add B_a upper to B_a lower and store the result in B_a lower.

7. Decrement the repeat count and test against 0. If unequal to 0, return to step 1.

Significantly, throughout the execution of the Block Transfer, the address contained in the u designator remains unchanged. Changes in B_b lower and B_a lower provide the different addresses to and from which data moves.

BLOCK TRANSFER

OPERATION CODE: 22

MNEMONIC CODE: BTR

OPERATION: (W)→(V). Repeat k times.

DESCRIPTION: Transfer the j-determined portion of the contents of W to the j-determined portion of V. Execute the instruction the number of times stipulated in the k portion of the repeat count register.

NOTES: See preceding paragraphs.

EXECUTION TIMES: 8.0. For practical purposes, there is no distinction between alternate banks and the same bank.

Because this instruction utilizes the repeat mode, an additional 12.0 microseconds are required for setup and termination of the repeat count.

12. SPECIAL INSTRUCTIONS

Three instructions in the UNIVAC 1107 repertoire are classified as special purpose instructions. In these instructions, the *j* designator serves as a minor operation code. The *a* designator is not utilized. The *b, h,* and *i* designators may be used to provide for index-register modification, incrementation of the modifier, and indirect addressing.

EXECUTE REMOTE INSTRUCTION

OPERATION CODE:	72
MINOR OPERATION CODE:	*j* = 10
MNEMONIC CODE:	EXRI

OPERATION: Execute the instruction stored at *U*

NOTES: Upon execution of the current instruction (operation code 72), *P* is not incremented.

Operations specified by the current instruction's *b, h,* and *i* designators are carried out before the remote instruction is read into *PCR*.

Remote instructions may be cascaded in the same manner as indirect addressing.

P is incremented upon execution of the final remote instruction.

EXECUTION TIME: 4.0, exclusive of the time required to execute the remote instruction.

LOAD MEMORY LOCKOUT REGISTER

OPERATION CODE:	72
MINOR OPERATION CODE:	*j* = 11
MNEMONIC CODE:	LMLR
OPERATION:	$U_{15-0} \rightarrow$ MLR

DESCRIPTION: Allow programmed writes to occur in certain areas of core memory while preventing its occurrence in other areas of core memory.

NOTES: The Memory Lockout instruction allows, in either or both core banks, the selective locking-in of groups of consecutive memory locations in increments of 2048 beginning at address 00000 or at any address which is a multiple of 2048.

The execution of the instruction causes the *U* portion of the instruction word, as modified by an index register (if called for), to be transferred to the Memory Lockout register. The two high-order bit positions of this 18-bit register are ignored.

The remaining 16 bits are divided into two groups of 8 bits each: group 1, bit positions 8-15, controls the selection of the addresses in core bank 1 which will be locked-in; group 2, bit positions 0-7, controls the selection of the addresses in core bank 2 which will be locked-in.

To lock-in an area in either bank it is necessary to indicate both a lower and an upper address which will be the start and end of the locked-in area.

	CORE 1		CORE 2	
XX	UPPER	LOWER	UPPER	LOWER
17 16	15 12	11 8	7 4	3 0

Consequently, each 8-bit group is subdivided into a lower section and an upper section. Each section contains 4 bit positions. The 4 bit positions of the lower sections are used to indicate the starting address for locked-in core and the 4 bit positions of the upper sections are used to indicate the ending address for locked-in core as seen in the following table.

ILLUSTRATIVE EXAMPLES:

1. MLR =

	CORE 1		CORE 2	
XX	0011	0000	1101	1001
17 16	15 12	11 8	7 4	3 0

CORE BANK 1: Since the lower value is equal to 0, the starting address of the locked-in core area is 00000. The upper value of 3 specifies an ending address of 081911 or the locked-in area. All other core addresses are locked-out against programming writes (as opposed to I/O writes which are never locked-out).

CORE BANK 2: The lower value of 9 indicates a starting address of 51200 and the upper value of 13 gives an ending address of 61439 for the locked-in area. Again, all other core addresses in this bank are protected against programmed writes.

SECTION VALUE		CORE BANK 1		SECTION VALUE		CORE BANK 2	
BINARY -	DECIMAL	LOWER	UPPER	BINARY -	DECIMAL	LOWER	UPPER
0000	0	00000	02047	0000	0	32768	34815
0001	1	02048	04095	0001	1	34816	36863
0010	2	04096	06143	0010	2	36864	38911
0011	3	06144	08191	0011	3	38912	40959
0100	4	08192	10239	0100	4	40960	43007
0101	5	10240	12287	0101	5	43008	45055
0110	6	12288	14335	0110	6	45056	47103
0111	7	14336	16383	0111	7	47104	49151
1000	8	16384	18431	1000	8	49152	51199
1001	9	18432	20479	1001	9	51200	53247
1010	10	20480	22527	1010	10	53248	55295
1011	11	22528	24575	1011	11	55296	57343
1100	12	24676	26623	1100	12	57344	59391
1101	13	26624	28671	1101	13	59392	61439
1110	14	28672	30719	1110	14	61440	63487
1111	15	30720	32767	1111	15	63488	65535

2. MLR =

	CORE 1		CORE 2	
xx	0000	0000	0000	0000
17 16	15 12	11 8	7 4	3 0

When the Memory Lockout register contains the above configuration, memory locations 00000-02047 of core bank 1 are locked-in and memory locations 32768-34815 of core bank 2 are locked-in. All other memory locations are locked-out against programmed writes.

3. MLR =

	CORE 1		CORE 2	
xx	1001	1000	0100	0000
17 16	15 12	11 8	7 4	3 0

When the Memory Lockout register contains the above configuration memory locations 16384-20479 of core bank 1 are locked-in and memory locations 32768-43007 of core bank 2 are locked-in.

4. MLR =

	CORE 1		CORE 2	
xx	0000	1111	1111	0000
17 16	15 12	11 8	7 4	3 0

Whenever the lower limit, for either core bank, has a value which is greater than its corresponding upper limit value the whole core bank is locked-out. Therefore, in this example all the locations of core bank 1 are locked-out. In order to lock-in all the locations of a core bank, the lower limit should be set equal to 0 and the upper limit should be set equal to 15. In this example all the locations in core bank 2 are locked-in.

The execution of a Memory Lockout instruction removes any previously set lockout.

NO OPERATION

OPERATION CODE: 74

MINOR OPERATION CODE: $j = 6$

MNEMONIC CODE: NOOP

OPERATION: Do nothing; continue with the next sequential instruction.

EXECUTION TIME: 4.0

13. FLOATING-POINT INSTRUCTIONS

Eight instructions in the UNIVAC 1107 Thin-Film Memory Computer repertoire provide for floating-point arithmetic. These instructions, used primarily in scientific computation, alert special circuitry built into the system. Ensuing floating-point arithmetic is then performed as a hardware function.

Data that will enter into floating-point calculations must adhere to the format of the floating-point word (see Figure 3-3). This format combines the mantissa, the characteristic, and the sign in a single word. The characteristic is biased by 128 (200 octal).

In floating-point instructions, the *a* designator always specifies an arithmetic register. Index register modification, incrementation of the modifier, and indirect addressing may be used in conjunction with every floating-point instruction. The *j* designator serves as a minor operation code.

Add, subtract, and multiply floating-point instructions always result in a 2-word answer, with the most significant word normalized. The least significant word is un-normalized. The quotient resulting from the divide instruction is always normalized while the remainder is retained in its original state.

Data that will enter into floating-point calculations need not be normalized. Division with un-normalized numbers may not produce logical results.

FLOATING ADD

OPERATION CODE: 76

MINOR OPERATION CODE: $j = 0$

MNEMONIC CODE: FLAD

OPERATION: $(A) + (U) \rightarrow A, A+1$

DESCRIPTION: Form the packed, normalized, floating-point sum of the numbers contained in *A* and *U*. Store the sum in *A* and *A + 1*.

EXAMPLE 1:		CHAR.	MANTISSA
where $(A)_i$	=	264	423456722
and (U)	=	250	663543211
THEN $(A)_f$	=	264	423545276
AND (A + 1)	=	231	321100000

NOTES: The mantissa of *U*, with the sign of the characteristic extended to the left, is sent to a working register in the arithmetic section. An adjacent register is filled with sign bits:

```
000663543211      000000000000
```

The absolute value of the characteristic of *U* is subtracted from the absolute value of the characteristic of *A*. Because the result is positive (+14), the mantissa of *U* is shifted right circularly by the difference (14 equals 12 places or 4 octal digit positions):

```
000000066354        321100000000
```

The mantissa of *A*, with the sign of the characteristic extended to the left, is added to the most significant word:

```
000000066354
000423456722
------------
000423545276
```

The least significant word is packed with the larger characteristic (264) minus the number of bit positions in the mantissa (always 33 octal), and stored in *A + 1:*

```
231       321100000 ——► A + 1
```

The most significant word is normalized and packed with the larger characteristic minus the number of positions shifted in normalizing. The result in stored in *A*. In the example, normalizing is not necessary (a binary 1 is already in bit position 26):

```
264       423545276 ——► A
```

EXAMPLE 2:

		CHAR.	MANTISSA
where $(A)_i$	=	253	403217654
and (U)	=	527	023775245
THEN $(A)_f$	=	252	613437001
AND $(A + 1)_f$	=	577	377777777

NOTES: See preceding notes.

Because the absolute value of the characteristic of *U* was arrived at via complementation, the characteristic of the least significant word in the result must be complemented before it is packed.

When there is a negative difference between characteristics, the mantissa of *A*, rather than *U*, is shifted right circularly. The mantissa of *U* is then added to the most significant word.

When the difference between characteristics is 0, mantissas are not shifted prior to addition.

A characteristic less than 0 in the least significant word will cause an interrupt to location 305. A characteristic greater than 377 in the most significant word will cause an interrupt to location 306. In either case, the original contents of *A* and *A + 1* are undisturbed.

EXECUTION TIMES:		
	Alternate Banks	14.0
	Same Bank	18.0

FLOATING SUBTRACT

OPERATION CODE: 76

MINOR OPERATION CODE: $j = 1$

MNEMONIC CODE: FLSB

OPERATION: (A) – (U)——►A, A+ 1

DESCRIPTION: Subtract the floating-point number in *U* from the floating-point number in *A*. Store the result in *A* and *A + 1*.

EXAMPLE:

		CHAR.	MANTISSA
where $(A)_i$	=	275	660000011
and $(U)_i$	=	250	420000002
THEN $(A)_f$	=	275	657777747
AND $(A + 1)_f$	=	242	000000200

NOTES: The mantissa of *U*, with the sign of the characteristic extended to the left, is sent to a working register in the arithmetic section. An adjacent register is filled with sign bits:

```
000 420000002       000 000000000
```

The absolute value of the characteristic of *U* is subtracted from the absolute value of the characteristic of *A*. Because the result is positive (+25), the mantissa of *U* is shifted right circularly by the difference:

```
000 000000042       000 000000200
```

The most significant half of the mantissa of *U* is subtracted from the mantissa of *A:*

```
000 660000011
000 000000042
-------------
000 657777747
```

The most significant half is normalized and packed with the larger characteristic minus the number of positions shifted in normalizing.

The result is stored in *A* (in the example, the result is already normalized):

```
275 657777747→A
```

The least significant word is packed with the larger characteristic minus the number of bit positions in the mantissa. The result is stored in *A + 1*.

```
242 000000200→A + 1
```

If the original characteristic of *U* had been larger than that of *A*, the mantissa of *U* would have been complemented prior to the shifting.

The interrupt conditions for Floating Subtract are the same as those for the Floating Add.

EXECUTION TIMES:	Alternate Banks	14.0
	Same Bank	18.0

FLOATING MULTIPLY

OPERATION CODE:	76
MINOR OPERATION CODE:	$j = 2$
MNEMONIC CODE:	FLMP
OPERATION:	$(A) \cdot (U) \rightarrow A, A+1$

DESCRIPTION: Multiply the floating-point number contained in *A* by the floating-point number contained in *U*. Store the packed floating-point product in *A* and *A + 1*.

EXAMPLE:		CHAR.	MANTISSA
where $(A)_i$	=	174	600000007
and $(U)_i$	=	220	500000006
THEN$(A)_f$	=	213	740000021
AND$(A + 1)_f$	=	160	600000124

NOTES: The absolute values of the mantissas of *A* and *U*, with zeros replacing their characteristics, are multiplied:

```
              000600000007
              000500000006
000000360000  010700000052
```

To conform to the floating-point format, the double-length product is shifted circularly 27 places to the left:

```
000360000010     700000052000
```

Counting right from the binary point (which lies to the immediate left of bit position 26 in the most significant word), it is noted that multiplication produced a 53-bit result. Consequently, a left circular shift of one is performed (a 54-bit product is not shifted more than 1 place):

```
000740000021     600000124000
```

The bias is subtracted from the sum of the characteristics. The difference, minus 1 (accruing from the left shift), is the characteristic of the most significant word. After packing, this word is stored in *A:*

```
174(Char.of A)   414          214
220(Char.of U)  -200(bias)    -1 (leftshift)
414              214          213
```

```
213 740000021→A
```

The characteristic of the most significant word, minus the number of bit positions in the mantissa (always 33 octal), is the characteristic of the least significant word. After packing, this word is stored in *A + 1:*

```
213(Char.of A)
-33
160                160 600000124→A + 1
```

If the signs of the original floating-point values had been different, the double-length product would have been complemented.

Multiplication of 27-bit normalized mantissas will always result in a product of 53 or 54 bits.

Multiplication of two un-normalized values may produce an un-normalized result.

EXECUTION TIMES:	Alternate Banks	13.3
	Same Bank	17.3

FLOATING DIVIDE

OPERATION CODE:	76
MINOR OPERATION CODE:	$j = 3$

MNEMONIC CODE: FLDV

OPERATION: (A) – (U)→A,A+1

DESCRIPTION: Divide the floating-point number contained in *A* by the floating-point number contained in *U*. Store the packed, floating-point quotient in *A* and the remainder in *A + 1*.

EXAMPLE:

	CHAR.	MANTISSA
where $(A)_i$	= 174	600000007
and $(U)_i$	= 150	400000006
THEN $(A)_f$	= 225	577777776
AND $(A + 1)_f$	= 171	000000030

NOTES: Division is performed by trial subtractions (dividend minus the divisor). If the absolute value of the mantissa of *U* is greater than or equal to the absolute value of the mantissa of *A*, 27 subtractions and shifts are performed; otherwise, 28 subtractions and shifts are performed.

In the subtraction * 0's replace the characteristics of both *A* and *U*. After the first subtraction, the remainder is shifted one place to the left while a binary 1 is inserted in the rightmost digit position in a working register:

```
000 000 000 110 000 000 000 000 000 000 000 111 A
000 000 000 100 000 000 000 000 000 000 000 110 U
            010                             001 remainder
            100                             010 left shift
```

Since the second subtraction will not take, a 0 is placed to the immediate right of the binary 1 previously inserted in the working register. The previous remainder is again shifted one place to the left:

```
000 000 000 100 000 000 000 000 000 000 000 010
000 000 000 100 000 000 000 000 000 000 000 110
              (subtraction does not take)
          1 000                             100 left shift
```

After the 27th subtraction and shift, the working register contains the quotient.

```
000  577777776
```

The remainder, after the final subtraction and shift, is:

```
000  000000030
```

The characteristic of the quotient is determined in the following manner: the characteristic of the dividend (174) minus the characteristic of the divisor (150) plus bias (200). If 28 subtractions had been made, this characteristic would be reduced by 1. After packing, the quotient is stored in *A*.

The characteristic of the remainder is the characteristic of the dividend minus 27 or 28, depending upon the number of trial subtractions. After packing, the remainder is stored in the next higher arithmetic register (*A + 1*).

If the dividend was negative, the remainder is complemented. If the signs of the original dividend and divisor were different, the quotient is complemented.

If the characteristic of the quotient exceeds 377, an overflow interrupt to location 306 occurs. If the characteristic of the remainder is less than 0, an underflow interrupt to location 305 occurs.

A divide overflow, that is, an interrupt to location 307, occurs only when the divisor is plus or minus 0.

EXECUTION TIMES:		
	Alternate Banks	26.7
	Same Bank	30.7

FLOATING POINT UNPACK

OPERATION CODE: 76

MINOR OPERATION CODE: $j = 4$

MNEMONIC CODE: FLUP

OPERATION: Unpack the floating-point number contained in *U*. Store the mantissa in *A + 1* and the biased characteristic in *A*.

EXAMPLE:

	CHAR.	MANTISSA
where (U)	= 264	423456722
THEN (A + 1)	= 000	423456722
AND (A)	= 000	000000264
where (U)	= 527	613437002
THEN (A + 1)	= 777	613437002
AND (A)	= 000	000000250

NOTES: The absolute value of the characteristic of *U* is stored in bit positions 0 through 7 in *A*, with 0's filled in to the left.

* *For clarity, the examples and trial subtractions are shown in binary, rather than octal.*

The mantissa of *U* is stored in *A + 1,* with sign bits in the 9 most significant bit positions.

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

FLOATING POINT NORMALIZE PACK

OPERATION CODE: 76

MINOR OPERATION CODE: $j = 5$

MNEMONIC CODE: FLNP

OPERATION: Form the packed, normalized, floating-point number from the mantissa stored in *U* and the characteristic stored in the low-order bit positions in *A*. Store the result in *A + 1*.

EXAMPLES:

			CHAR.	MANTISSA
1. where	(U)	=	000	423456722
and	(A)	=	000	000000264
THEN	(A+1)	=	264	423456722
2. where	(U)	=	777	613437002
and	(A)	=	000	000000250
THEN	(A+1)	=	527	613437002
3. where	(U)	=	777	045667432
and	(A)	=	000	000000155
THEN	(A+1)	=	622	456674320

NOTES: The upper 28 bit positions in *A* are ignored.

The characteristic in *A* is adjusted according to the required normalization.

Before storing it in *A + 1,* the characteristic, after normalization, is complemented when the the sign of the mantissa in *U* is negative.

A characteristic overflow will cause an interrupt to location 306. A characteristic underflow will cause an interrupt to location 305. In either case, *A + 1* is undisturbed.

EXECUTION TIMES: Alternate Banks 7.3
Same Bank 11.3

FLOATING CHARACTERISTIC DIFFERENCE MAGNITUDE

OPERATION CODE: 76

MINOR OPERATION CODE: $j = 6$

MNEMONIC CODE: FLCM

OPERATION: Absolute value $|(A)_{34-27}| - |(U)_{34-27}| \longrightarrow A + 1$

DESCRIPTION: Subtract the absolute value of the characteristic of *U* from the absolute value of the characteristic of *A*. Store the absolute value of the difference in low-order positions in *A + 1*.

EXAMPLES:

			CHAR.	MANTISSA
1. where	(A)	=	264	423456722
and	(U)	=	250	663543211
THEN	(A+1)	=	000	000000014
2. where	(A)	=	253	463217654
and	(U)	=	527	023775245
THEN	(A+1)	=	000	000000003
3. where	(A)	=	250	663543211
and	(U)	=	264	423456722
THEN	(A+1)	=	000	000000014

NOTES: The absolute value of the difference is stored in low-order positions in *A + 1,* with 0's filled in to the left.

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

FLOATING CHARACTERISTIC DIFFERENCE

OPERATION CODE: 76

MINOR OPERATION CODE: $j = 7$

MNEMONIC CODE: FLCD

OPERATIONS: $|(A)_{34-27}| - |(U)_{34-27}| \rightarrow A + 1$

DESCRIPTION: Subtract the absolute value of the characteristic of *U* from the absolute value of characteristic of *A*. Store the difference in low-order positions in *A + 1*.

EXAMPLES:

			CHAR.	MANTISSA
1. where	(A)	=	250	663543211
and	(U)	=	264	423456722
THEN	(A+1)	=	777	777777763

NOTES: The difference is stored in low-order posiin *A + 1,* with sign bits filled into the left.

EXECUTION TIMES: Alternate Banks 4.0
Same Bank 8.0

14. CONTROL CONSOLE

The Control Console includes the operator's control panel, a keyboard and type-printer, and a control unit for the keyboard and type-printer. Optionally, a paper-tape reader and a paper-tape punch can be connected to the Computer through the same control unit.

By means of this arrangement, data may enter the system via the keyboard or the paper-tape reader. Direct communication from the Computer is made via the type-printer or the paper-tape punch. The type-printer permits spot-checking of the program currently being processed.

Two switches mounted on the control unit permit manual selection of the keyboard (or the paper-tape reader) and the typeprinter (or the paper-tape punch). Selection of a given unit can also be made through program control.

The input-output units associated with the Control Console will be discussed in detail in the manual covering peripheral equipment.

Operator's Control Panel

This panel provides direct operator communication with the Computer. The manual controls and indicators allow the operator to perform the following operations:

1. Stop program execution while allowing input-output operations to continue.

2. Clear all registers except those in the input-output section.

3. Master clear all registers including those in the input-output section.

4. Set the desired starting (or restarting) address into the *P*-register. Six octal-digit indicators display the contents of the *P*-register.

5. Start program execution. The program will start with the execution of the instruction located at the address contained in the *P*-register.

6. Set any of fifteen *selective jump* switches. An indicator is lit when the corresponding jump is selected. A jump can be selected while a program is running.

7. Set any of four *selective stop* switches. When the selection is made, the upper half of a corresponding indicator is lit. The lower half of this indicator is lit when the appropriate stop is made. A stop can be selected while a program is running.

8. Select one of sixteen channels for the initial loading of a bootstrap routine.

9. Read (Load) the bootstrap routine.

Four fault or status indicators are also included on the operator's control panel. These indicators, along with examples of the type of conditions they reveal, are presented below.

Computer Status – excessive temperature; poor voltage regulation.

Program Faults – illegal operation code; illegal memory address (an address within a locked-out memory area).

Peripheral Equipment Fault – loss of power in a channel synchronizer; disconnected cable.

Initial Loading (Bootstrap Fault) – error occuring during the loading of the bootstrap program.

AUTOMATIC PROGRAMMING

The Automatic Programming Library for the UNIVAC 1107 System will include the following programs:

- **COBOL** – A data-processing compiler. The specifications for Basic COBOL were defined in a Department of Defense publication dated April, 1960. COBOL 1961, which incorporates significant improvements, serves as the basis for this compiler.

- **ALGOL** – An algebraic language compiler. The specifications for this compiling system were developed jointly by the Association of Computing Machinery (ACM) Committee on Programming Languages and the GAMM* Committee on Programming. The report was published in the *Communications of the ACM*, May and July, 1960.

- **FORTRAN** – A translator that will accept problems written in FORTRAN II Language. This routine will enable problems previously coded in FORTRAN to be run on the UNIVAC 1107 Thin-Film Memory Computer without revision.

- **SIMULATOR** – A routine that will interpretatively execute the instruction repertoire of the UNIVAC 1107 System on a UNIVAC 1103-A, 1103-AS, or 1105 Computer. By means of this routine, programs written for the UNIVAC 1107 may be run and corrected, if necessary, before the Computer itself is available.

- **BASIC UTILITY LIBRARY** – A library of routines coded expressly for the UNIVAC 1107 Thin-Film Memory Computer. The following programs will be included in the utility library:

  - **ASSEMBLY SYSTEM.** An advanced computer-oriented mnemonic code assembly system will be provided. This routine will accept instructions containing symbolic operand addresses and mnemonic function codes and designators. It will then translate these instructions into an absolute or relative form, ready for loading into the Computer. The assembler will also provide the means for correcting source code, allocating assembled programs, producing parallel output of source and assembled programs, and incorporating library routines.

  - **EXECUTIVE SYSTEM.** A routine that will automatically accomplish the execution of runs in compliance with a predetermined Computer schedule. In this capacity, the executive routine will extract the programs that are to be executed, position them in their operating locations, and provide for the time-sharing of several programs running in parallel. This routine will also incorporate special checking features for the problem run.

  - **SORT-MERGE PROGRAM.** A set of routines to arrange random items in an ordered sequence. Routines will also be available for combining two or more ordered sequences into a single file (on the basis of information contained in specified fields of each item).

  - **INPUT-OUTPUT ROUTINES.** A set of routines to perform the input and output functions for standard peripheral equipment.

  - **DEBUGGING AIDS.** A set of routines to aid the programmer in checking out a particular program.

  - **FUNCTION EVALUATION ROUTINES.** A set of commonly used mathematical routines. The initial set will include sine, cosine, tangent, arc sine, arc cosine, arc tangent, square root, natural logarithm, and exponential. These routines will be compatible with fixed-point and floating-point arithmetic.

  - **LIBRARIAN ROUTINE.** A routine for building and maintaining a library of subroutines. It will be capable of inserting, deleting, or changing routines in the library, as well as extracting routines for use in a particular program. With this routine, the library may be altered at will to conform to individual customer requirements.

**Gesellschaft fur Angewandte Mathematik und Mechanik.*

APPENDIX A. INSTRUCTION REPERTOIRE

f	j	NAME	DESCRIPTION	EXECUTION TIME IN μ SEC.		MNEMONIC CODE
				Alternate Core Banks	Same Core Bank	
01	0-17	Store Positive	$(A) \rightarrow U$	4.0	8.0	STP
02		Store Negative	$-(A) \rightarrow U$	4.0	8.0	STN
03		Store Magnitude	$\|(A)\| \rightarrow U$	4.0	8.0	STM
04		Store R_a	$(R_a) \rightarrow U$	4.0	8.0	STR
05		Store Zero	$0 \rightarrow U$ (Clear U)	4.0	8.0	STZ
06		Store B_a	$(B_a) \rightarrow U$	4.0	8.0	STB
10		Load Positive	$(U) \rightarrow A$	4.0	8.0	LDP
11		Load Negative	$-(U) \rightarrow A$	4.0	8.0	LDN
12		Load Positive Magnitude	$\|(U)\| \rightarrow A$	4.0	8.0	LDM
13		Load Negative Magnitude	$-\|(U)\| \rightarrow A$	4.0	8.0	LNM
14		Add	$(A) + (U) \rightarrow A$	4.0	8.0	ADD
15		Subtract	$(A) - (U) \rightarrow A$	4.0	8.0	SUB
16		Add Magnitude	$(A) + \|(U)\| \rightarrow A$	4.0	8.0	ADM
17		Subtract Magnitude	$(A) - \|(U)\| \rightarrow A$	4.0	8.0	SBM
20		Add and Load	$(A) + (U) \rightarrow A + 1$	4.0	8.0	ADL
21		Subtract and Load	$(A) - (U) \rightarrow A + 1$	4.0	8.0	SBL
22†		Block Transfer	$(W)_i \rightarrow (V)_i$ repeated k times. Initial V_1 address is $u + (B_b)_{17-0}$, and subsequent addresses are formed by incrementation by $(B_b)_{35-18}$. Similarly, V_2 addresses are $u + (B_a)_{17-0}$ incremented by $(B_a)_{35-18}$.	8.0	8.0	BTR
23		Load R_a	$(U) \rightarrow R_a$	4.0	8.0	LDR
24		Add to B_a	$(B_a) + (U) \rightarrow B_a$	4.0	8.0	ADB
25		Subtract from B_a	$(B_a) - (U) \rightarrow B_a$	4.0	8.0	SBB
26		Load B_a Modifier Only	$(U) \rightarrow B_{a17-0}$	4.0	8.0	LBM
27		Load B_a	$(U) \rightarrow B_a$	4.0	8.0	LDB
30		Multiply Integer	$(A) \bullet (U) \rightarrow A, A + 1$	12.0	16.0	MPI
31		Multiply Single (Integer)	$(A) \bullet (U) \rightarrow A$	12.0	16.0	MPS
32		Multiply Fractional	$(A) \bullet (U) \rightarrow A, A + 1$	12.0	16.0	MPF
34		Divide (Integer)	$(A, A + 1) \div (U)$; Quotient $\rightarrow A$ Remainder $\rightarrow A + 1$	31.3	35.3	DVI
35		Divide Single and Load (Fractional)	$(A) \div (U)$; Quotient $\rightarrow A + 1$ No Remainder	31.3	35.3	DVL
36		Divide (Fractional)	$(A, A + 1) \div (U)$; Quotient $\rightarrow A$ Remainder $\rightarrow A + 1$	31.3	35.3	DVF
40		Selective Set	$(A) \rightarrow A + 1$. Then set $(A + 1)_n$ for $(U)_n = 1$ i.e., $(A) \oplus (U) \rightarrow A + 1$	4.0	8.0	SSE
41		Selective Complement	$(A) \rightarrow A + 1$. Then complement $(A + 1)_n$ for $(U)_n = 1$ i.e., $(A) \overline{\oplus} (U) \rightarrow A + 1$	4.0	8.0	SCP
42		Selective Clear	$(A) \rightarrow A + 1$. Then clear $(A + 1)_n$ for $(U)_n = 1$ i.e., $(A) \odot (U) \rightarrow A + 1$	4.0	8.0	SCL
43		Selective Substitute	$(A) \rightarrow A + 1$. Then $(U)_n \rightarrow (A + 1)_n$ for $(M)_n = 1$ i.e., $(A) \odot (M)' + (U) \odot (M) \rightarrow A + 1$	4.7	8.7	SSU
44		Selective Even Parity Test	If $[(A) \odot (U)]$ is even parity, Skip NI No Skip Skip	 6.0 10.0	 10.0 14.0	SEP
45		Selective Odd Parity Test	If $[(A) \odot (U)]$ is odd parity, Skip NI No Skip Skip	 6.0 10.0	 10.0 14.0	SOP
47		Test Modifier	If $(B_a)_{17-0} < (U)$, take NI; If $(B_a)_{17-0} > (U)$, Skip. In either case, $(B_a)_{17-0} + (B_a)_{35-18} \rightarrow B_{a17-0}$ No Skip Skip	 4.7 8.7	 8.7 12.7	TMO
50		Test Zero	Skip NI if $(U) = 0$ No Skip Skip	 4.0 8.0	 8.0 12.0	TZR
51		Test Not Zero	Skip NI if $(U) \neq 0$ No Skip Skip	 4.0 8.0	 8.0 12.0	TNZ
52		Test Equal	Skip NI if $(U) = (A)$ No Skip Skip	 4.0 8.0	 8.0 12.0	TEQ
53		Test Not Equal	Skip NI if $(U) \neq (A)$ No Skip Skip	 4.0 8.0	 8.0 12.0	TNE
54		Test Less Than or Equal	Skip NI if $(U) \leq (A)$ No Skip Skip	 4.0 8.0	 8.0 12.0	TLE
55		Test Greater Than	Skip NI if $(U) > (A)$ No Skip Skip	 4.0 8.0	 8.0 12.0	TGR
56		Test Within Limits	Skip NI if $(A) < (U) \leq (A + 1)$ (Note: $(A) < (A + 1)$) No Skip Skip	 4.7 8.7	 8.7 12.7	TWL
57		Test Outside Limits	Skip NI if $(U) \leq (A)$ or $(U) > (A + 1)$ (Note: $(A) < (A + 1)$) No Skip Skip	 4.7 8.7	 8.7 12.7	TOL

† Repeat operations 62-67, 71 take 16 μ sec combined setup and termination time. The block transfer (22) takes 12 μ sec combined setup and termination time.

INSTRUCTION REPERTOIRE

f	j	NAME	DESCRIPTION		EXECUTION TIME IN μ SEC.		MNEMONIC CODE
					Alternate Core Banks	Same Core Bank	
60	0-17	Test Positive	Skip NI if $(U) \geq 0$	No Skip Skip	4.0 8.0	8.0 12.0	TPO
61		Test Negative	Skip NI if $(U) < 0$	No Skip Skip	4.0 8.0	8.0 12.0	TNG
62†		Search Equal	Skip NI if $(U)_i = (A)$ Repeated k times	No Skip Skip	4.0 4.0	4.0 4.0	SEQ
63†		Search Not Equal	Skip NI if $(U)_i \neq (A)$ Repeated k times	No Skip Skip	4.0 4.0	4.0 4.0	SNE
64†		Search Less Than or Equal	Skip NI if $(U)_i \leq (A)$ Repeated k times	No Skip Skip	4.0 4.0	4.0 4.0	SLE
65†		Search Greater Than	Skip NI if $(U)_i > (A)$	No Skip Skip	4.0 4.0	4.0 4.0	SGR
66†		Search Within Limits	Skip NI if $(A) < (U)_i \leq (A+1)$ (Note: $(A) < (A+1)$)	No Skip Skip	4.7 4.7	4.7 4.7	SWL
67†		Search Outside Limits	Skip NI if $(U)_i \leq (A)$ or $(U)_i > (A+1)$ (Note: $(A) < (A+1)$)	No Skip Skip	4.7 4.7	4.7 4.7	SOL
70		Index Jump	If $(CM)_{ja} > 0$, Jump to U $(CM)_{ja} \leq 0$, Take NI Then $(CM)_{ja} - 1 \rightarrow CM_{ja}$ NOTE: j in this instruction serves with the a-designator to specify any one of the 128 words of Control Memory.	No Jump Jump	8.0 4.0	8.0 4.0	IXJP
71†	*						
	00	Masked Search Equal	Skip NI if $(U)_i \odot (M) = (A) \odot (M)$ Repeated k times	No Skip Skip	4.0 4.0	4.0 4.0	MSEQ
	01	Masked Search Not Equal	Skip NI if $(U)_i \odot (M) \neq (A) \odot (M)$ Repeated k times	No Skip Skip	4.0 4.0	4.0 4.0	MSNE
	02	Masked Search Less Than or Equal	Skip NI if $(U)_i \odot (M) \leq (A) \odot (M)$ Repeated k times	No Skip Skip	4.0 4.0	4.0 4.0	MSLE
	03	Masked Search Greater Than	Skip NI if $(U)_i \odot (M) > (A) \odot (M)$ Repeated k times	No Skip Skip	4.0 4.0	4.0 4.0	MSGR
	04	Masked Search Within Limits	Skip NI if $(A) \odot (M) < (U)_i \odot (M)$ $\leq (A+1) \odot (M)$ (Note: $(A) \odot (M) <$ $(A+1) \odot (M)$) Repeated k times	No Skip Skip	4.7 4.7	4.7 4.7	MSWL
	05	Masked Search Outside Limits	Skip NI if $(U)_i \odot (M) \leq (A)$ or $(U) \odot (M) < (A+1)$ (Note: $(A) \odot (M) <$ $(A+1) \odot (M)$) Repeated k times	No Skip Skip	4.7 4.7	4.7 4.7	MSOL
72	*						
	00	Wait for Interrupt	The computer program sequence stops (i.e., P is not advanced). The wait condition is removed by an interrupt.		4.0		WAIT
	01	Return Jump	$(P) \rightarrow U_{17\text{--}0}$ and Jump to $U+1$		8.0	8.0	RTJP
	02	Positive Bit Control Jump	If $(A)_{35} = 0$, Jump to U Shift (A) left one in either case	No Jump Jump	4.0 8.0	4.0 8.0	PBJP
	03	Negative Bit Control Jump	If $(A)_{35} = 1$, Jump to U Shift (A) left one in either case	No Jump Jump	4.0 8.0	4.0 8.0	NBJP
	04	Add Halves	$(A)_{17\text{--}0} + (U)_{17\text{--}0} \rightarrow A_{17\text{--}0}$ $(A)_{35\text{--}18} + (U)_{35\text{--}18} \rightarrow A_{35\text{--}18}$		4.0	8.0	ADDH
	05	Subtract Halves	$(A)_{17\text{--}0} - (U)_{17\text{--}0} \rightarrow A_{17\text{--}0}$ $(A)_{35\text{--}18} - (U)_{35\text{--}18} \rightarrow A_{35\text{--}18}$		4.0	8.0	SUBH
	06	Add Thirds	$(A)_{35\text{--}24} + (U)_{35\text{--}24} \rightarrow A_{35\text{--}24}$ $(A)_{23\text{--}12} + (U)_{23\text{--}12} \rightarrow A_{23\text{--}12}$ $(A)_{11\text{--}0} + (U)_{11\text{--}0} \rightarrow A_{11\text{--}0}$		4.0	8.0	ADDT
	07	Subtract Thirds	$(A)_{35\text{--}24} - (U)_{35\text{--}24} \rightarrow A_{35\text{--}24}$ $(A)_{23\text{--}12} - (U)_{23\text{--}12} \rightarrow A_{23\text{--}12}$ $(A)_{11\text{--}0} - (U)_{11\text{--}0} \rightarrow A_{11\text{--}0}$		4.0	8.0	SUBT
	10	Execute Remote Instruction	Execute the Instruction at U		4.0	— + Execution Time	EXRI
	11	Load Memory Lockout Register	$U_{5\text{--}0} \rightarrow MLR$ For $U_0 = 1$ lockout 0—4095 $U_1 = 1$ lockout 4096—8191 $U_2 = 1$ lockout 8192—16383 $U_3 = 1$ lockout 16384—32767 $U_4 = 1$ lockout applies to 1st BANK $U_5 = 1$ lockout applies to 2nd BANK		4.0	—	LMLR
73‡	*						
	00	Single Right Circular Shift‡	Shift (A) right U places circularly		4.0		SCSH
	01	Double Right Circular Shift	Shift (A, A + 1) right U places circularly		4.0		DCSH
	02	Single Right Logical Shift	Shift (A) right U places, end off; fill with zeros (Max. Shift — 36)		4.0		SLSH

*j serves as part of the Function Code

† Repeat operations 62-67, 71 take 16 μ sec combined setup and termination time. The block transfer (22) takes 12 μ sec combined setup and termination time.

‡Instruction execution time is independent of the number of shifts performed (e.g. a shift of 72 takes 4 microseconds). There are no memory references in the first six shift instructions, 73 00 — 73 05; consequently, the distinction between **alternate** core banks and the **same** core bank is irrelevant.

INSTRUCTION REPERTOIRE

	j	NAME	DESCRIPTION	EXECUTION TIME IN μ SEC.		MNEMONIC CODE
				Alternate Core Banks	Same Core Bank	
	03	Double Right Logical Shift	Shift (A, A + 1) right U places, end off; fill with zeros. (Max. Shift = 72)	4.0		DLSH
	04	Single Right Arithmetic Shift	Shift (A) right U places, end off; fill with sign bits.	4.0		SASH
	05	Double Right Arithmetic Shift	Shift (A, A + 1) right U places, end off; fill with sign bits. (Max. Shift = 72)	4.0		DASH
	06	Scale Factor Shift	(U) → A, shift A left circularly until $A_{35} \neq A_{34}$ or until A has been shifted 36 times. Store the scaled quantity in A and the number of shifts that occurred in A + 1.	6.0	10.0	SFSH
74	*					
	00	Zero Jump	Jump to U if (A) = 0 No Jump Jump	4.0 8.0	4.0 8.0	ZRJP
	01	Non-zero Jump	Jump to U if (A) ≠ 0 No Jump Jump	4.0 8.0	4.0 8.0	NZJP
	02	Positive Jump	Jump to U if (A) ≥ 0 No Jump Jump	4.0 8.0	4.0 8.0	POJP
	03	Negative Jump	Jump to U if (A) < 0 No Jump Jump	4.0 8.0	4.0 8.0	NGJP
	04	Console Selective Jump	Jump to U if A = key setting on console (1 of 15)	4.0	4.0	CSJP
	05	Selective Stop Jump	Stop if A = stop key setting on console (1 of 4), always jump to U	4.0	4.0	SSJP
	06	No Operation	Do Nothing; continue with NI	4.0	4.0	NOOP
	07	Enable All External Interrupts and Jump	Jump to U and permit interrupts to occur	4.0	4.0	EIJP
	10	Even Jump	Jump to U if $(A)_0 = 0$ No Jump Jump	4.0 8.0	4.0 8.0	EVJP
	11	Odd Jump	Jump to U if $(A)_0 = 1$ No Jump Jump	4.0 8.0	4.0 8.0	ODJP
	12	Modifier Jump	If $(B_a)_{17\text{--}0} > 0$, Jump to U No Jump If $(B_a)_{17\text{--}0} < 0$, Take NI Jump In either case $(B_a)_{17\text{--}0} + (B_a)_{35\text{--}18} \rightarrow B_{a17\text{--}0}$	4.0 8.0	4.0 8.0	MOJP
	13	Load Modifier and Jump	(P) → $(B_a)_{17\text{--}0}$ and Jump to U	4.0	4.0	LMJP
	14	Overflow Jump	Jump to U if overflow cond. is set	4.0	4.0	OVJP
	15	No-Overflow Jump	Jump to U if overflow cond. is not set	4.0	4.0	NOJP
	16	Carry Jump	Jump to U if carry cond. is set	4.0	4.0	CYJP
	17	No-Carry Jump	Jump to U if carry cond. is not set	4.0	4.0	NCJP
75	*					
	00	Initiate Input Mode	(U) → input control word a, and initiate input mode on channel a.	4.0	8.0	IIPM
	01	Initiate Monitored Input Mode	(U) → input control word a, and initiate input mode on channel a with monitor.	4.0	8.0	IMIM
	02	Input Mode Jump	Jump to U if channel a is in the input mode.	4.0	4.0	IMJP
	03	Terminate Input Mode	Terminate input mode on channel a.	4.0	4.0	TIPM
	04	Initiate Output Mode	(U) → output control word a, and initiate output mode on channel a.	4.0	8.0	IOPM
	05	Initiate Monitored Output Mode	(U) → output control word a, and initiate output mode on channel a with monitor.	4.0	8.0	IMOM
	06	Output Mode Jump	Jump to U if channel a is in the output mode.	4.0	4.0	OMJP
	07	Terminate Output Mode	Terminate output mode on channel a.	4.0	4.0	TOPM
	10	Initiate Function Mode	(U) → output control word a, and initiate function mode on channel a.	4.0	8.0	IFNM
	11	Initiate Monitored Function Mode	(U) → output control word a, and initiate function mode on channel a with monitor.	4.0	8.0	IMFM
	12	Function Mode Jump	Jump to U if channel a is in the function mode.	4.0	4.0	FMJP
	13	Force External Transfer	Request external function or output word on channel a.	4.0	4.0	FEXT
	14	Enable All External Interrupts	All external interrupts are permitted to occur.	4.0	4.0	EAEI
	15	Disable All External Interrupts	All external interrupts are prevented from occurring.	4.0	4.0	DAEI
	16	Enable Single External Interrupt	An external interrupt on channel a is permitted to occur.	4.0	4.0	ESEI
	17	Disable Single External Interrupt	An external interrupt on channel a is prevented from occurring.	4.0	4.0	DSEI
76	*					
	00	Floating Add	(A) + (U) → A, A + 1	14.0	18.0	FLAD
	01	Floating Subtract	(A) — (U) → A, A + 1	14.0	18.0	FLSB
	02	Floating Multiply	(A) • (U) → A, A + 1	13.3	17.3	FLMP
	03	Floating Divide	(A) ÷ (U); Quotient → A Remainder → A + 1	26.7	30.7	FLDV
	04	Floating Point Unpack	Unpack (U), store mantissa in A + 1 and store the biased characteristic in A	4.0	8.0	FLUP
	05	Floating Point Normalize Pack	Normalize (A) pack with biased characteristic from (U) and store at A + 1	7.3	11.3	FLNP
	06	Floating Characteristic Difference Magnitude	Absolute value of $\lvert(A)_{34\text{--}27}\rvert - \lvert(U)_{34\text{--}27}\rvert \rightarrow A + 1$	4.0	8.0	FLCM
	07	Floating Characteristic Difference	$\lvert(A)_{34\text{--}27}\rvert - \lvert(U)_{34\text{--}27}\rvert \rightarrow A + 1$	4.0	8.0	FLCD

Remington Rand Univac

DIVISION OF SPERRY RAND CORPORATION

PRINTED IN U.S.A. UT2463