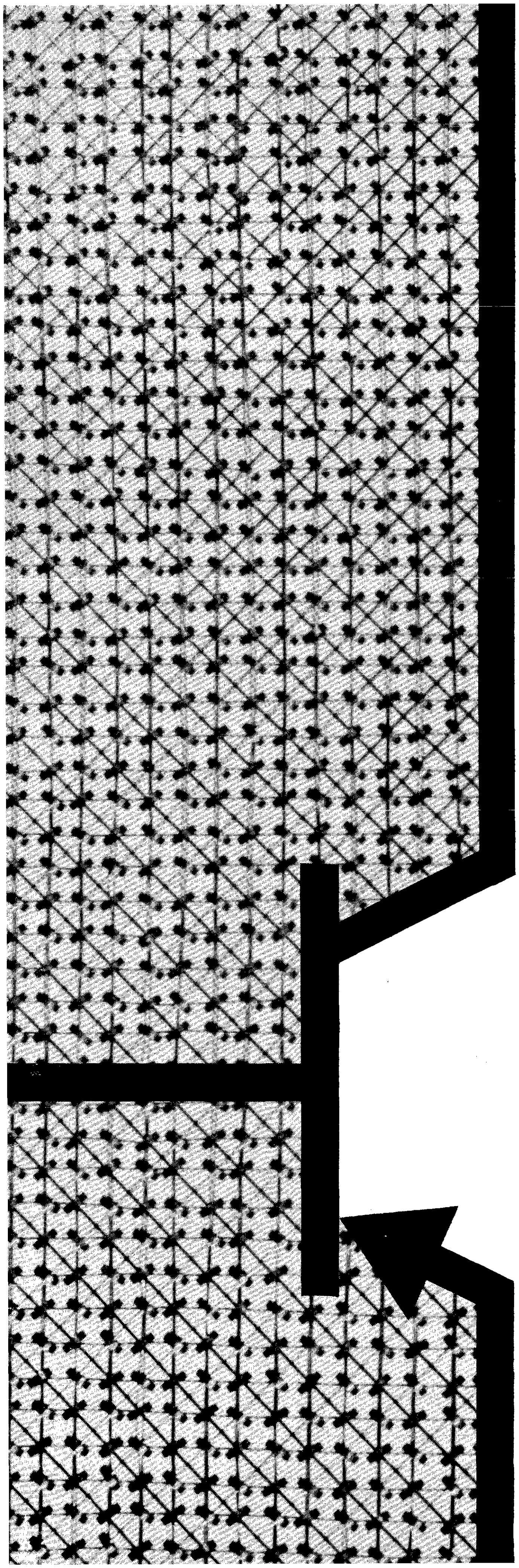
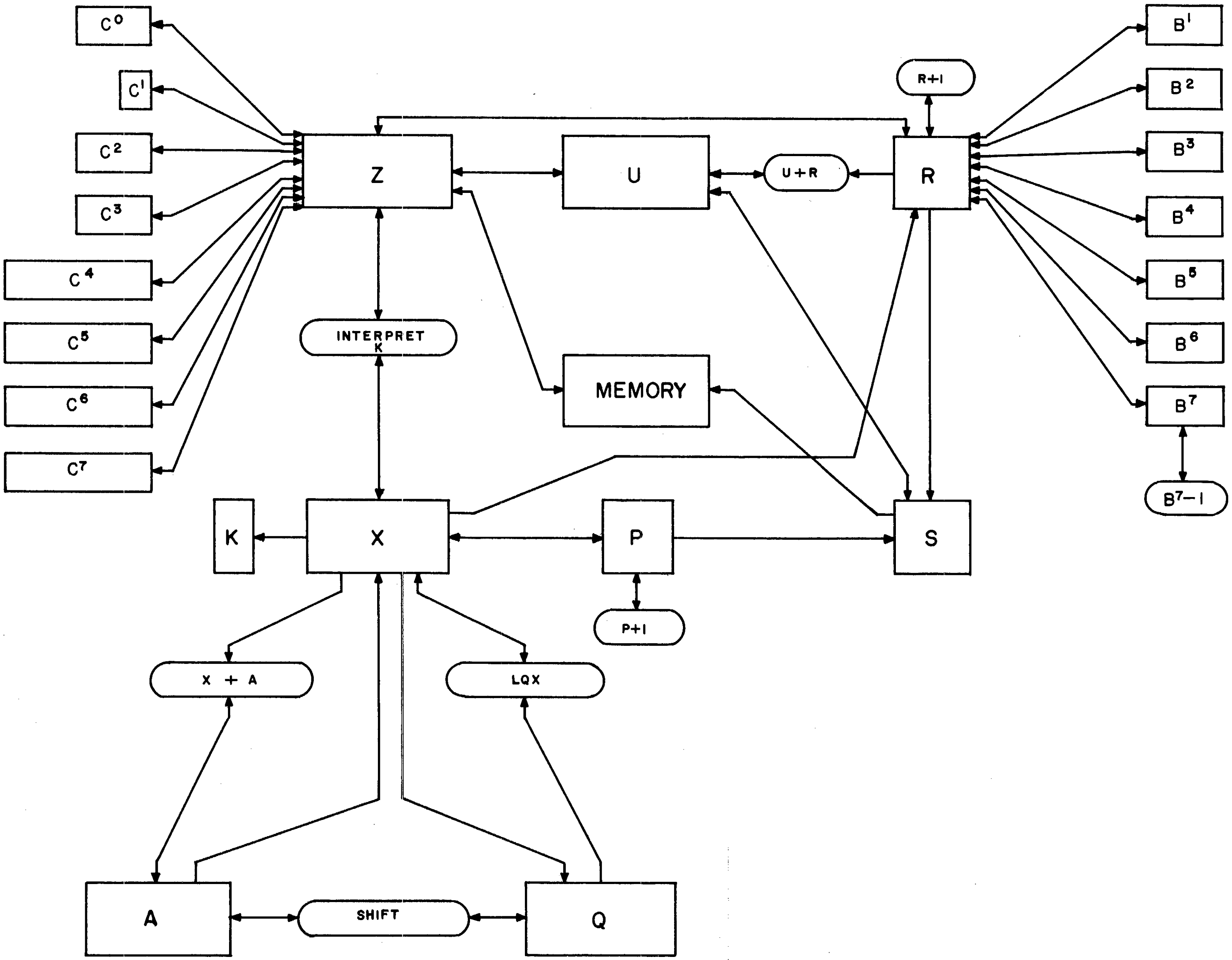


M-460 COMPUTER Characteristics

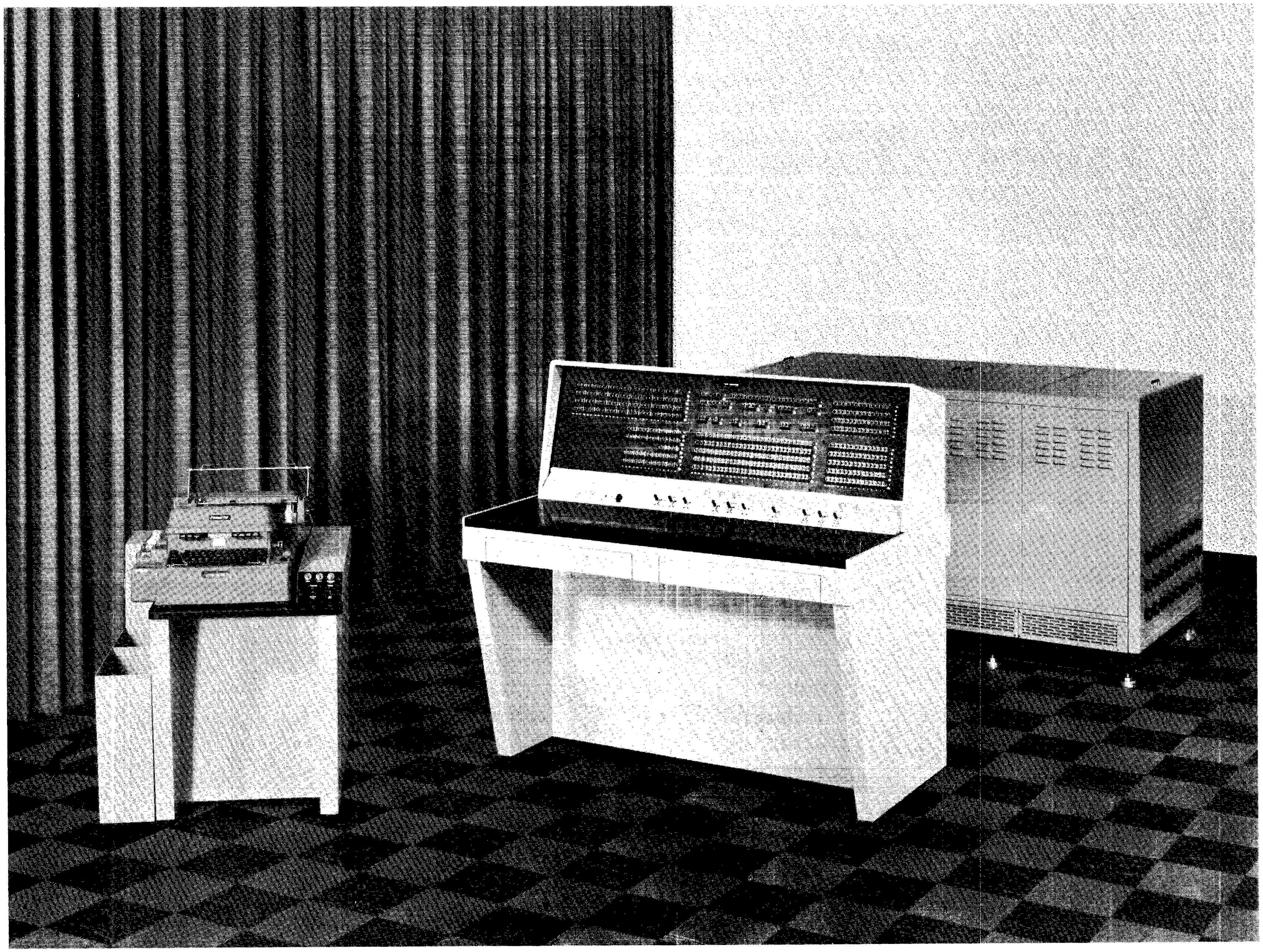
Remington Rand Univac®

DIVISION OF SPERRY RAND CORPORATION

UNIVAC PARK · SAINT PAUL 16, MINNESOTA



Simplified Logical Diagram of the Univac M-460 Computer.



The Univac M-460 Computer.

M-460
COMPUTER CHARACTERISTICS

GENERAL

The Univac M-460 Computer is a stored program computer intended for the rapid handling of large quantities of complex data. Relative to other general purpose computer systems, the M-460 Computer emphasizes random access storage and communication with external devices. The internal operations are performed in a parallel binary mode, with a 30-bit instruction word and either a 15- or 30-bit data word. Instructions are of the one-address type, with an average execution time of 20 microseconds. Synchronous logic is used with a 2 megacycle clock rate.

The internal storage of the computer consists of a maximum of 8 modular units. Each of these units consists of 4096 magnetic core storage locations with a 30-bit capacity per location. Each of the possible 32,768 storage locations may be interpreted as a single 30-bit word, or as two 15-bit words individually addressed. The control, arithmetic, and input-output sections of the computer have independent access to the storage section. Information flows over parallel transmission paths between the storage unit and other units in the system. A complete storage cycle of *read* followed by *write* requires 8 microseconds.

The arithmetic and logical operations are performed in a parallel binary mode. In general, the result appears in a 30-bit accumulator register. Arithmetic is *ones* complement subtractive with a modulus 2^{30}-1. An auxiliary arithmetic register, designated as the Q-register, is used in multiply, divide, and logical operations. The Q-register holds the multiplier at the beginning of a multiply operation, the least significant half of the double length product at the end of the operation. The Q-register also holds the least significant half of the dividend at the beginning of a divide operation, the quotient at the end of the operation.

Computer operation is controlled by a stored program capable of self-modification. Each program instruction contains a function code (6 bits), one storage address (15 bits), and three execution modifiers (3 bits each). The execution modifiers provide for address incrementation, operand interpretation, and branch point designation. A storage address may be incremented by any one of seven index registers. The operand specified by the execution address may be interpreted as a 30-bit quantity, or as a 15-bit half word with or without sign extension. The next sequential program step may be skipped under control of the contents of the arithmetic register.

Communication between the M-460 Computer and associated external equipment is normally handled by a block transfer of data, with the timing under the control of the external device. Operating asychronously with the main computer program, such transfers of data employ independent access to storage.

A communication path is established by a sequence of external request and response signals between the external equipment and the M-460 Computer. Such signals may originate either at the computer or at the external equipment. The main computer program is interrupted by external request signals and establishes the communications channel. Once the link is established, the computer returns to the main program sequence. The block transfer of input or output data proceeds without program reference until completed.

A total of 18 input and 12 output channels is provided in the computer. These channels vary in size from 6 to 30 parallel lines per channel. The 30 channels are divided into three groups, viz., input, output, and function lines. Buffering registers and control circuits are provided to permit concurrent activation of 7 channels. The maximum possible transfer rate of input or output data over a given channel is 1,500,000 bits per second.

The M-460 Computer is constructed primarily of diodes, transistors, and magnetic cores. The logical sections are implemented by transistor unit packages. However, the storage section contains combinations of transistors and magnetic core arrays. The basic computer is contained in a single main cabinet 3′ x 3′ x 6′. Total power consumption is 1200 watts. Forced air cooling is provided without a heat exchanger.

REGISTERS

The M-460 Computer contains a number of registers in which data is held during computation. These registers are designated by a letter or by a letter and a numeral, and are interconnected by parallel transmission paths over which information flows during processing. The registers are identified below in two categories, operational and transient. Operational registers hold information from one instruction to another, and are referenced in the operational description of each instruction. On the other hand, transient registers are temporary storage locations, which are always cleared at the end of an instruction.

OPERATIONAL REGISTERS

The **A-Register** (30 bits), or accumulator, is the principle arithmetic register. It is provided with parallel addition and shifting properties. In the majority of arithmetic instructions, the result of the operation is left in the A-register for use in later program steps. Thus, after addition or subtraction the sum or difference remains in the accumulator. However, after multiplication the most significant half of the product remains in the accumulator. After division, the remainder is left in the accumulator.

The contents of the A-register may be shifted either to the left or to the right as described in the shift instructions. When the shift is to the left, the transfer is circular. That is, the rightmost digits are replaced with the leftmost digits. When the shift is to the right, the sign bit is extended by the number of bit positions shifted; and the lower order digits are discarded.

On certain instructions, the A and Q registers are shifted as a single 60-bit register. In all such cases, the A-register represents the most significant half of the double length quantity.

The **Q-Register** (30-bits), or auxiliary arithmetic register, assists the accumulator in multiply, divide, and logical operations. The Q-register has shifting and logical properties, but it has no addition or counting functions.

The contents of the Q-register may be shifted to the right or left as a 30-bit register or as the lower order half of a 60-bit register in conjunction with the accumulator. With the exception of the shift paths, all communication is via the X-register. Logical multiplication is performed on a transmission path between the Q and X registers.

The Q-register holds the multiplier at the beginning of a multiply operation. As the product is formed by repeated additions and shifts, the multiplier digits are shifted to the right and discarded. In their place, the lower order digits of the double length product are shifted into the Q-register from the accumulator.

During a divide operation, a process essentially the reverse of multiplication takes place. The double length dividend is shifted to the left, and quotient bits are inserted in the rightmost position of the Q-register. At the end of the divide sequence, the quotient is assembled in the Q-register; and the remainder is left in the accumulator.

The **P-Register** (15-bits) is the program address counter. This register holds the address of the next sequential instruction throughout the program. As each program address is transferred from the P-register to the S-register, the contents of the P-register are increased by one. When executing jump instructions, the P-register is cleared; and a new program address is entered.

The **B-Registers** (15 bits each) are address modifying registers generally used for indexing minor loops in a program. The contents of one register may be used to increment the operand address before execution of an instruction. A total of seven such registers are provided, with two of these serving additional control functions as described in instructions 70, 73, and 74.

The **C-Registers** (6 to 30 bits each) are communication buffer registers through which input and output data are synchronized. A total of eight C registers are provided. C-register 0 has a special function as described below. Each of the other seven registers serves a group of input and output channels.

C-Register 0 (15 bits) is a communication channel switching register, which by its content controls the channel selection for the other seven registers. The various bit positions in this register may be interpreted in seven groups: three bits for C-register 3, and two bits for each of the other C registers. The various binary combinations in each group then determine the channel selection for the corresponding C-register.

C-Register 1 (6 bits) is an input-output register used to communicate monitoring information. The monitoring typewriter and keyboard entry communication are via this register. In addition, perforated paper tape may also be processed through this register.

C-Register 2 (15 bits) normally is assigned to input and output devices requiring relatively few bits of information per message. Such devices as analog-to-digital converters fall in this category.

C-Register 3 (15 bits) normally is assigned as an external function request and response register. Computer function requests are sent to external equipment via this register, and response signals return via this register.

C-Registers 4, 5, 6, and 7 (30 bits each) are the principle input-output registers. Information may be transferred through these registers over 30 parallel lines. Communication with displays, magnetic tapes, magnetic drums, other computers, etc., is handled through these registers.

TRANSIENT REGISTERS

The following registers are used in the manipulation of instruction words and data words during the execution of an instruction. These registers are not referenced in the description of the instructions, and do not retain information from one operation to the next.

The **X-Register** (30 bits) functions as an arithmetic communication register. It has complementing, but no shift properties. The X-register holds the operand from storage during all arithmetic operations. All communication between the A and Q registers and the rest of the operational registers is via the X-register.

The **K-Register** (6 bits) functions as a shift counter for all arithmetic operations involving shifts. The maximum shift count is 64. Multiply and divide operations are controlled by pre-setting the K-register to 30, and by counting the operational steps.

The **S-Register** (15 bits) holds the storage address during memory references. At the beginning of a storage access period, the address is transferred to the S-register. The contents of the S-register are then translated to activate the storage selection system.

The **Z-Register** (30 bits) serves as an operand buffer for storage references. During the *read* portion of the storage access period, the Z-register is cleared. The digit-reading amplifiers are then sampled to set the contents of Z corresponding to bits in storage. During the *write* portion of the storage access period, the Z-register controls the inhibit circuits to write or restore the disturbed storage register.

The **U-Register** (30 bits) is the program control register. In other words, it holds the instruction word during the execution of an operation. The operation code and the various execution modifiers are translated from appropriate sections of the register. The lower order 15 bits of the U-register have addition properties, modulus 2^{15}-1. If an address modification is required before execution, the contents of the appropriate B-register are added to the contents of the lower order 15 bits of the U-register before execution.

The **R-Register** (15 bits) functions as a communication register for the B registers. All internal transmissions to or from the B registers pass through the R-register. It also holds the incrementing quantity during address modification. This register has complementing and counting provisions for incrementing the contents of the B-register.

INSTRUCTION FORMAT

Each step in the operation of the M-460 Computer is controlled by an instruction in a stored program. An instruction is a 30-bit word which is read from magnetic core storage at the appropriate time and entered in the U-register, where it controls the particular operation required. Various portions of an instruction word are interpreted for various aspects of the required operation while the instruction resides in the U-register. These portions, or designators, are identified by a lower case letter for reference in the description of the instructions. They are listed below as they appear from left to right in the U-register.

f—function code designator (6 bits)
j—branch condition designator (3 bits)
k—operand interpretation designator (3 bits)
b—address modification designator (3 bits)
y—operand address designator (15 bits)

Normally, these designators become effective in the reverse of the order listed.

- The first action following the entry of an instruction in the U-register is the modification of **y**—the operand address designator—by the contents of a B-register as specified by the **b** designator. This modification consists of adding the contents of the B-register to the **y** designator.
- The second action is normally a storage reference to obtain the quantity at the resultant storage address, and to enter it in the Z-register.
- The third action is a transmission of the contents of the Z-register to the X-register, with the character of the transmission under the control of the **k** designator.
- Subsequent action affects the contents of the operational register as specified by the **f** designator, and procures the next instruction as specified by the **j** designator.

The quantity which enters the X-register in the above process is called the operand. Since it is dependent on a number of designators, the operand is identified by a symbol, (Y), in describing the various instructions. The storage location from which the operand comes is identified as Y. A letter in parenthesis associated with an operational register is used as a shorthand symbol for the contents of the operational register. However, this relationship is not quite true in the case of the letter Y, since the **k** designator may distort the quantity which comes from address Y before it becomes the operand (Y). Summarizing the shorthand symbols,

Y—the storage location of the operand
(Y)—the operand
(A)—the contents of the accumulator
(Q)—the contents of the Q-register
(B2)—the contents of B-register 2
(C5)—the contents of C-register 5

Function Code Designator—f (6 bits) The highest order six bits of an instruction designate the function to be performed by that instruction. All values other than 00 and 77 are defined in the instruction list. The two codes 00 and 77 are fault conditions, which if executed will cause the computation to stop. The fault light will then be illuminated on the operation control panel.

Branch Condition Designator—j (3 bits) This octal designator normally specifies the condition under which the next sequential instruction in the program will be skipped. This provides for branching from a sequence without executing the jump instruction, if the branch is not taken. Particular significance is associated with this designator in the case of repeated instructions. In such cases, the exit from the repeat mode is under the control of the **j** designator. A skip of the next sequential instruction is determined by the following rules, except where otherwise noted in the instruction list.

j=0	do not skip the next instruction
j=1	skip the next instruction
j=2	skip the next instruction if (Q) is positive
j=3	skip the next instruction if (Q) is negative
j=4	skip the next instruction if (A) is zero
j=5	skip the next instruction if (A) is non-zero
j=6	skip the next instruction if (A) is positive
j=7	skip the next instruction if (A) is negative

Operand Interpretation Designator—k (3 bits) The operand interpretation designator controls the transmission of the operand from the Z-register to the X-register, and vice versa.

Those instructions which read an operand but do not replace it after the arithmetic is performed are designated *read* instructions. Those instructions that do not read an operand but store one are designated *store* instructions. Instructions which both read and write operands are known as *replace* instructions.

For *read* instructions, the **k** designator controls the transmission of the operand from the Z-register to the X-register. This transmission may select only half of the contents of the 30-bit Z-register, and may or may not extend the resulting upper bit throughout the higher order 15 bits of the X-register. This provision allows a half word to be treated either as a 15-bit positive number or as a 14-bit number plus sign. In the case of **k**=0, 4, or 7, the lower order 15 bits of the U-register are not used to specify a storage address for the procurement of the operand. For **k**=0 and **k**=4, the lower order 15 bits of the U-register are used directly as a 15-bit operand. In the case of **k**=7, the contents of the accumulator are used as a 30-bit operand.

The effect of the various values of the **k** designator on the operand are listed below. This applies to all *read* instructions, except where noted under the individual instruction.

For *read* instructions:

k=0	Clear the X-register. Then transmit the lower order 15 bits of (U) to the lower order 15 bits of X.
k=1	Clear the X-register. Then transmit the lower order 15 bits of (Z) to the lower order 15 bits of X.
k=2	Clear the X-register. Then transmit the higher order 15 bits of (Z) to the lower order 15 bits of X.

k=3	Clear the X-register. Then transmit the entire 30 bits of (Z) to X.
k=4	Clear the X-register. Then transmit the lower order 15 bits of (U) to the lower order 15 bits of X. If bit position 14 of (X) is *one*, set each of the higher 15 bits of X to *one*.
k=5	Clear the X-register. Then transmit the lower order 15 bits of (Z) to the lower order 15 bits of X. If bit position 14 of (X) is *one*, set each of the higher order 15 bits of X to *one*.
k=6	Clear the X-register. Then transmit the higher order 15 bits of (Z) to the lower order 15 bits of X. If bit position 14 of (X) is *one*, set each of the higher order 15 bits of X to *one*.
k=7	Clear the X-register. Then transmit (A) to all 30 bits of the X-register.

The operand interpretation designator **k** controls the transmission from the X-register to the Z-register on *write* operations. The effect of the **k** designator in *store* instructions is defined below, except where noted under the individual instructions.

For *store* instructions:

k=0	Clear the Q-register. Then transmit (X) to the Q-register.
k=1	Replace the lower order 15 bits of the quantity stored at address Y with the lower order 15 bits of (X)—leaving the higher order 15 bits undisturbed.
k=2	Replace the higher order 15 bits of the quantity stored at the address Y with the lower order 15 bits of (X)—leaving the lower order 15 bits undisturbed.
k=3	Replace the quantity stored at address Y with (X).
k=4	Clear the accumulator. Then add (X) to the accumulator.
k=5	Replace the lower order 15 bits of the quantity stored at address Y with the complement of the lower order 15 bits of (X)—leaving the higher order 15 bits undisturbed.
k=6	Replace the higher order 15 bits of the quantity stored at address Y with the complement of the lower order 15 bits of (X)—leaving the lower order 15 bits undisturbed.
k=7	Replace the quantity stored at address Y with the complement of (X).

The *replace* instructions combine a reading operation with a writing operation. For the read portion the operand interpretation designator **k** has the same meaning as the *read* instructions. For the write portions, a different interpretation is used.

For *replace* instructions:

k=0	Not used.
k=1	During the read portion, clear the X-register. Then transmit the lower order 15 bits of (Z) to the lower order 15 bits of X. During the write portion, replace the lower order 15 bits of the quantity stored at address Y with the lower order 15 bits of (X)—leaving the higher order 15 bits undisturbed.
k=2	During the read portion, clear the X-register. Then transmit the higher order 15 bits of (Z) to the lower order 15 bits of X. During the write portion, replace the higher order 15 bits of the quantity stored at address Y with the lower order 15 bits of (X)—leaving the lower order 15 bits undisturbed.
k=3	During the read portions, clear the X-register. Then transmit the entire 30 bits of (Z) to X. During the write portion, replace the quantity stored at address Y with (X).
k=4	Not used.
k=5	During the *read* portion, clear the X-register. Then transmit the lower order 15 bits of (Z) to the lower order 15 bits of X. If bit position 14 of (X) is *one*, set each of the higher order 15 bits of X to *one*. During the write portion, replace the lower order 15 bits of the quantity stored at address Y with the lower order 15 bits of (X)—leaving the higher order 15 bits undisturbed.
k=6	During the read portion, clear the X-register. Then transmit the higher order 15 bits of (Z) to the lower order 15 bits of X. If bit position 14 of (X) is *one*, set each of the higher order 15 bits of X to *one*. During the write portion, replace the higher order 15 bits of the quantity stored at address Y with the lower order 15 bits of (X)—leaving the lower order 15 bits undisturbed.
k=7	Not used.

Address Modification Designator—b (3 bits) The address modification designator specifies which of the B registers, if any, will be used to modify the operand address designator **y** before a storage reference is made. If a modification takes place, the contents of a B-register is added to the lower order 15 bits of (U). This operation employs an additive accumulator. Hence, the quantity consisting of all *zeros* cannot occur as a result, unless both the operand address designator **y** and the contents of the B-register are all *zeros*. The effect of the various values of the **b** designator are

b=0	Do not modify **y**.
b=0	Add the contents of selected B-register to **y**.

Operand Address Designator—y (15 bits) The operand address designator specifies, subject to modification, which of the 32,768 storage locations will be referenced in the execution of an instruction. During execution, most instructions reference this storage location once. In some cases, however, the operand is read from storage, operated upon, and then returned to the same storage location. Such instructions are described as *replace* instructions in the listing. Other instructions, such as those with **k** = 0 or **k** = 4, make no storage reference for an operand—but take the operand directly from the lower 15 bits of the instruction word itself.

INSTRUCTION LIST FOR THE UNIVAC M-460 COMPUTER

00	not used	40	enter logical product
01	shift Q right	41	add logical product
02	shift A right	42	subtract logical product
03	shift AQ right	43	masked comparison
04	compare	44	replace logical product
05	shift Q left	45	replace add logical product
06	shift A left	46	replace subtract logical product
07	shift AQ left	47	store logical product
10	enter Q-register	50	selective set
11	enter accumulator	51	selective complement
12	enter B-register	52	selective clear
13	enter C-register	53	substitute
14	store Q-register	54	replace selective set
15	store accumulator	55	replace selective complement
16	store B-register	56	replace selective clear
17	store C-register	57	replace substitute
20	add	60	arithmetic jump
21	subtract	61	manual jump
22	multiply	62	input jump
23	divide	63	output jump
24	add replace	64	arithmetic return jump
25	subtract replace	65	manual return jump
26	Q add	66	input return jump
27	Q subtract	67	output return jump
30	load A, add Q	70	initiate repeat
31	load A, subtract Q	71	index skip
32	add Q and store	72	index jump
33	subtract Q and store	73	initiate input transfer
34	replace add Q	74	initiate output transfer
35	replace subtract Q	75	initiate input buffer
36	replace add one	76	initiate output buffer
37	replace subtract one	77	not used

DETAILED EXPLANATION OF INSTRUCTION REPERTOIRE

(01) Shift Q Right—This instruction shifts (Q) to the right (Y) bit positions. The higher order bits are replaced with the original sign bit as the word is shifted. Only the lower order six bits of (Y) are recognized for this instruction.

(02) Shift A Right—This instruction shifts (A) to the right (Y) bit positions. The higher order bits are replaced with the original sign bit as the word is shifted. Only the lower order six bits of (Y) are recognized for this instruction.

(03) Shift AQ Right—This instruction shifts (A) and (Q) as one 60-bit register. The shift is to the right (Y) bit positions, with the lower bits of (A) shifting into the higher bits of (Q). The higher order bits of (A) are replaced with the original sign bit as the word is shifted. Only the lowest order six bits of (Y) are recognized for this instruction.

(04) Compare—This instruction compares (Y) with (A) and/or (Q). It does not alter either (A) or (Q). The branch condition designator is interpreted in a special way for this instruction.

j = 0	do not skip the next instruction.
j = 1	skip the next instruction.
j = 2	skip the next instruction if (Y) is less than, or equal to, (Q).
j = 3	skip the next instruction if (Y) is greater than (Q).
j = 4	skip the next instruction if (Q) is greater than, or equal to, (Y) and (Y) is greater than (A).
j = 5	skip the next instruction if (Y) is greater than (Q) or if (Y) is less than, or equal to, (A).
j = 6	skip the next instruction if (Y) is less than, or equal to, (A).
j = 7	skip the next instruction if (Y) is greater than (A).

(05) Shift Q Left—This instruction shifts (Q) circularly to the left (Y) bit positions. The lower order bits are replaced with the higher order bits as the word is shifted. Only the lower order six bits of (Y) are recognized for this instruction.

(06) Shift A Left—This instruction shifts (A) circularly to the left (Y) bit positions. The lower order bits are replaced with the higher order bits as the word is shifted. Only the lower order six bits of (Y) are recognized for this instruction.

(07) Shift AQ Left—This instruction shifts (A) and (Q) as one 60-bit register. The shift is circular to the left (Y) bit positions. The lower bits of (A) are replaced with the higher bits of (Q), and vice versa. Only the lower order six bits of (Y) are recognized by this instruction.

(10) Enter Q-Register—Clear the Q register. Then transmit (Y) to Q.

(11) Enter Accumulator—Clear the accumulator. Then add (Y) to the accumulator.

(12) Enter B-Register—Clear the contents of B-register **j**. Then transmit the lower order 15 bits of (Y) to B-register **j**. The higher order 15 bits of (Y) are ignored in this instruction. The branch condition designator **j** is used to specify the selected B-register for this instruction and is not available for its normal function.

(13) Enter C-Register—Clear the contents of C-register **j**. Then transmit (Y) to C-register **j**. If the selected C-register is not a 30-bit register, then the lower order portion of (Y) is transmitted, the higher order portion ignored. Finally, set the selected C-register indicator bit to *one*. The branch condition designator **j** is used to specify the selected C-register for this instruction and is not available for its normal function.

(14) Store Q-Register—Store (Q) at storage address Y as directed by the operand interpretation designator **k**. For $\mathbf{k} = 0$, clear the Q-register. Then transmit the complement of (X) to the Q-register, which has the effect of complementing (Q).

(15) Store Accumulator—Store (A) at storage address Y as directed by the operand interpretation designator **k**. For $\mathbf{k} = 4$, clear the accumulator. Then transmit the complement of (X) to the accumulator, which has the effect of complementing (A).

(16) Store B-Register—Form in the X-register a 30-bit quantity, the lower order 15 bits of which correspond to the contents of B-register **j**, the higher order 15 bits of which are *zero*. Then store (X) at storage address Y as directed by the operand interpretation designator **k**. The branch condition designator **j** is used to specify the selected B-register for this instruction and is not available for its normal function.

(17) Store C-Register—Store the contents of C-register **j** at storage address Y as directed by the operand interpretation designator **k**. If the selected C-register is not a 30-bit register, then the higher order bits are interpreted as *zeros*. Clear the selected C-register indicator bit, but not the register. The branch condition designator **j** is used to specify the selected C-register for this instruction and is not available for its normal function.

(20) Add—Add (Y) to the previous contents of the accumulator.

(21) Subtract—Subtract (Y) from the previous contents of the accumulator.

(22) Multiply—Multiply (Q) times (Y), leaving the double length product in AQ. If the factors are considered as integers, the product is an integer in AQ. The branch condition designator **j** is interpreted prior to end correction, thus permitting overflow detection by sensing the sign of the accumulator.

(23) Divide—Divide (AQ) by (Y), leaving the quotient in the Q-register, the remainder in the accumulator. The remainder bears the same sign as the quotient. The branch condition designator **j** is interpreted prior to end correction, thus permitting overflow detection by sensing the sign of the accumulator.

(24) Add Replace—Add (Y) to the previous contents of the accumulator. Then store (A) at the storage address Y as directed by the operand interpretation designator **k.**

(25) Subtract Replace—Subtract (Y) from the previous contents of the accumulator. Then store (A) at the storage address Y as directed by the operand interpretation designator **k.**

(26) Q Add—Shift left 30 places (A) and (Q) as a single 60-bit register. Then add (Y) to the accumulator. Next, shift left (A) and (Q) 30 places as a single 60-bit register. The contents of the accumulator are undisturbed by this instruction. The branch condition designator **j** has special meaning in this instruction:

j = 0	do not skip the next instruction
j = 1	skip the next instruction
j = 2	skip the next instruction if (A) is positive
j = 3	skip the next instruction if (A) is negative
j = 4	skip the next instruction if (Q) is zero
j = 5	skip the next instruction if (Q) is non zero
j = 6	skip the next instruction if (Q) is positive
j = 7	skip the next instruction if (Q) is negative

(27) Q Subtract—Shift left 30 places (A) and (Q) as a single 60-bit register. Then subtract (Y) from the accumulator. Next, shift left (A) and (Q) 30 places as a single 60-bit register. The contents of the accumulator are undisturbed by this instruction. The branch condition designator **j** has special meaning in this instruction as listed under (26) Q Add.

(30) Load A Add Q—Clear the accumulator, add (Q) to the accumulator, and then add (Y) to the accumulator.

(31) Load A Subtract Q—Clear the accumulator, subtract (Q) from the accumulator, and add (Y) to the accumulator.

(32) Add Q and Store—Add (Q) to the previous contents of the accumulator; store (A) at storage address Y as directed by the operand interpretation designator **k**. The branch condition designator **j** is not interpreted for values 0 and 4 of the operand interpretation designator **k.**

(33) Subtract Q and Store—Subtract (Q) from the previous contents of the accumulator. Then store (A) at storage address Y as directed by the operand interpretation designator **k**. The branch condition designator **j** is not interpreted for values 0 and 4 of the operand interpretation designator **k.**

(34) Replace Add Q—Clear the accumulator, add (Q) to the accumulator, add (Y) to the accumulator, and then store (A) at storage address Y as directed by the operand interpretation designator **k.**

(35) Replace Subtract Q—Clear the accumulator. Subtract (Q) from the accumulator. Next, add (Y) to the accumulator. Then store (A) at storage address Y as directed by the operand interpretation designator **k.**

(36) Replace Add One—Clear the accumulator. Add one to the accumulator. Then store (A) at storage address Y as directed by the operand interpretation designator **k.**

(37) Replace Subtract One—Clear the accumulator. Add minus one to the accumulator. Next, add (Y) to the accumulator. Then store (A) at storage address Y as directed by the operand interpretation designator **k.**

(40) Enter Logical Product—Clear the accumulator. Form in the X-register the bit-by-bit product of (Y) and (Q). Then add (X) to the accumulator.

(41) Add Logical Product—Form in the X-register the bit-by-bit product of (Y) and (Q). Then add (X) to the previous contents of the accumulator.

(42) Subtract Logical Product—Form in the X-register the bit-by-bit product of (Y) and (Q). Then subtract (X) from the previous contents of the accumulator.

(43) Masked Comparison—Form in the X-register the bit-by-bit product of (Y) and (Q). Next, subtract (X) from the previous contents of the accumulator. Then perform the branch point evaluation for skipping the next sequential instruction as directed by the branch condition designator **j**. Finally, add (X) to the accumulator.

This instruction results in no net change in the contents of any operational register. Through the branch condition designator, it provides a comparison of a portion of (Y) with the contents of the accumulator.

(44) Replace Logical Product—Clear the accumulator. Form in the X-register the bit-by-bit product of (Y) and (Q). Next, add (X) to the accumulator. Then store (A) at storage address Y as directed by the operand interpretation designator **k.**

(45) Replace Add Logical Product—Form in the X-register the bit-by-bit product of (Y) and (Q). Next, add (X) to the previous contents of the accumulator. Then store (A) at storage address Y as directed by the operand interpretation designator **k**.

(46) Replace Subtract Logical Product—Form in the X-register the bit-by-bit product of (Y) and (Q). Next, subtract (X) from the previous contents of the accumulator. Then store (A) at storage address Y as directed by the operand interpretation designator **k**.

(47) Store Logical Product—Form in the X-register the bit-by-bit product of (A) and (Q). Then store (X) at storage address Y as directed by the operand interpretation designator **k**. The branch condition designator **j** is not interpreted for values 0 and 4 of the operand interpretation designator **k**.

(50) Selective Set—Set individual bits of (A) to *one* corresponding to *ones* in (Y), leaving the remaining bits of (A) unaltered.

(51) Selective Complement—Complement individual bits of (A) corresponding to *ones* in (Y), leaving the remaining bits of (A) unaltered.

(52) Selective Clear—Clear individual bits of (A) corresponding to *ones* in (Y), leaving the remaining bits of (A) unaltered.

(53) Substitute—Clear individual bits of (A) corresponding to *ones* in (Q), leaving the remaining bits of (A) unaltered. Next, form in the X-register the bit-by-bit product of (Y) and (Q). Then set individual bits of (A) to *one*, corresponding to *ones* in (X)—leaving the remaining bits of (A) unaltered.

This instruction has the effect of replacing bits of (A) with bits of (Y) corresponding to *ones* in (Q).

(54) Replace Selective Set—Set individual bits of (A) to *one* corresponding to *ones* in (Y), leaving the remaining bits of (A) unaltered. Then store (A) at storage address Y as directed by the operand interpretation designator **k**.

(55) Replace Selective Complement—Complement individual bits of (A) corresponding to *ones* in (Y), leaving the remaining bits of (A) unaltered. Then store (A) at storage address Y as directed by the operand interpretation designator **k**.

(56) Replace Selective Clear—Clear individual bits of (A) corresponding to *ones* in (Y), leaving the remaining bits of (A) unaltered. Then store (A) at storage address Y as directed by the operand interpretation designator **k**.

(57) Replace Substitute—Clear individual bits of (A) corresponding to *ones* in (Q), leaving the remaining bits of (A) unaltered. Form in the X-register the bit-by-bit product of (Y) and (Q). Next, set individual bits of (A) to *one* corresponding to *ones* in (X), leaving the remaining bits of (A) unaltered. Then store (A) at storage address Y as directed by the operand interpretation designator **k**.

This instruction has the effect of replacing bits of (Y) with bits of (A) corresponding to *zeros* in (Q).

(60) Arithmetic Jump—This instruction clears the program address register, P, and enters a new program address for certain conditions of the contents of arithmetic registers. The branch condition designator **j** is interpreted in a special way for this instruction, determining the conditions under which a jump in a program address occurs.

j = 0	No action. Continue with the current program sequence.
j = 1	Execute jump.
j = 2	Execute jump if (Q) is positive.
j = 3	Execute jump if (Q) is negative.
j = 4	Execute jump if (A) is zero.
j = 5	Execute jump if (A) is non zero.
j = 6	Execute jump if (A) is positive.
j = 7	Execute jump if (A) is negative.

If the jump condition is not satisfied, the next sequential instruction in the current sequence is executed in a normal manner. If the jump condition is satisfied, then the lower order 15 bits of (Y) become the address of the next instruction, and the beginning of a new program sequence. The higher order 15 bits of (Y) are not used in this instruction.

(61) Manual Jump—This instruction clears the program address register, P, and enters a new program address for certain conditions of manual key selections. The branch condition designator **j** is interpreted in a special way for this instruction, determining the conditions under which a jump in a program address occurs.

j = 0	Execute jump regardless of key selections.
j = 1	Execute jump if Key 1 is selected.
j = 2	Execute jump is Key 2 is selected.
j = 3	Execute jump if Key 3 is selected.
j = 4	Execute jump. Then stop computation.
j = 5	Execute jump. Then stop computation if Key 5 is selected.
j = 6	Execute jump. Then stop computation if Key 6 is selected.
j = 7	Execute jump. Then stop computation if Key 7 is selected.

If the jump condition is not satisfied, the next sequential instruction in the current sequence is executed in a normal manner. If the jump condition is satisfied, then the lower order 15 bits of (Y) become the address of the next instruction, and the beginning of a new program sequence. The higher order 15 bits of (Y) are not used in this instruction.

The program may be stopped by certain key selections upon the execution of this instruction. The branch condition designator **j** specifies which key selections are effective.

(62) Input Jump—This instruction clears the program address register, P, and enters a new program address for certain conditions determined by the status of the C-register. The branch condition designator **j** is interpreted in a special way for this instruction to designate which C-register status will be examined. If the C-register **j** indicator bit is in the *one* state, indicating information in the register, then the jump condition is satisfied. If the selected indicator bit is in the *zero* state, then the jump condition is not satisfied. If the jump condition is not satisfied, the next sequential instruction in the current sequence is executed in a normal manner. If the jump condition is satisfied, then the lower order 15 bits of (Y) become the address of the next instruction, and the beginning of a new program sequence. The higher order 15 bits of (Y) are not used in this instruction.

(63) Output Jump—This instruction clears the program address register, P, and enters a new program address for certain conditions determined by the status of the C-register. The branch condition designator **j** is interpreted in a special way for this instruction to designate which C-register status will be examined. If the C-register **j** indicator bit is in the *zero* state, indicating no information in the register, then the jump condition is satisfied. If the selected indicator bit is in the *one* state, then the jump condition is not satisfied. If the jump condition is not satisfied, the next sequential instruction in the current sequence is executed in a normal manner. If the jump condition is satisfied, then the lower order 15 bits of (Y) become the address of the next instruction, and the beginning of a new program sequence. The higher order 15 bits of (Y) are not used in this instruction.

(64) Arithmetic Return Jump—This instruction executes a return jump sequence for certain conditions determined by the contents of the arithmetic register. The branch condition designator **j** is interpreted in a special way for this instruction, determining the conditions under which the return jump sequence is executed.

j = 0	No action. Continue with the current program sequence.
j = 1	Execute return jump.
j = 2	Execute return jump if (Q) is positive.

j = 3 Execute return jump if (Q) is negative.
j = 4 Execute return jump if (A) is zero.
j = 5 Execute return jump if (A) is non zero.
j = 6 Execute return jump if (A) is positive.
j = 7 Execute return jump if (A) is negative.

If the return jump condition is not satisfied, then the next sequential instruction in the current sequence is executed in a normal manner. If the return jump condition is satisfied, shown above, then the following sequence is performed:

Clear the X-register and transmit (P) to the lower order 15 bits of X. Clear the P-register and transmit the lower order 15 bits of (Y) to P. Then store the lower order 15 bits of (X) in the lower order 15 bits of storage address (P), leaving the higher order 15 bits undisturbed. Finally, increase (P) by one for the program address of the next instruction.

(65) Manual Return Jump—This instruction executes a return jump sequence for conditions determined by manual key selections. The branch condition designator **j** is interpreted in a special way for this instruction, determining the conditions under which the return jump sequence is executed.

j = 0 Execute return jump regardless of key selections.
j = 1 Execute return jump if Key 1 is selected.
j = 2 Execute return jump if Key 2 is selected.
j = 3 Execute return jump if Key 3 is selected.
j = 4 Execute return jump. Then stop computation.
j = 5 Execute return jump. Then stop computation if Key 5 is selected.
j = 6 Execute return jump. Then stop computation if Key 6 is selected.
j = 7 Execute return jump. Then stop computation if Key 7 is selected.

If the return jump condition is not satisfied, then the next sequential instruction in the current sequence is executed in a normal manner. If the return jump condition is satisfied, then the following sequence is performed:

Clear the X-register and transmit (P) to the lower order 15 bits of X. Clear the P-register and transmit the lower order 15 bits of (Y) to P. Store the lower order 15 bits of (X) in the lower order 15 bits of storage address (P), leaving the higher order 15 bits undisturbed. Then increase (P) by one for the program address of the next instruction.

(66) Input Return Jump—This instruction executes a return jump sequence for certain conditions determined by the status of the C-register. The branch condition designator **j** is interpreted in a special way for this instruction to designate which C-register status will be examined. If the C-register **j** indicator bit is in the *one* state, indicating information in the register, then the return jump condition is satisfied.

If the selected indicator bit is in the *zero* state, then the return jump condition is not satisfied. If the return jump condition is not satisfied, the next sequential instruction in the current sequence is executed in a normal manner. If the return jump condition is satisfied, then the following sequence is performed:

Clear the X-register and transmit (P) to the lower order 15 bits of X. Clear the P-register and transmit the lower order 15 bits of (Y) to P. Then store the lower order 15 bits of (X) in the lower order 15 bits of storage address (P), leaving the higher order 15 bits undisturbed. Finally, increase (P) by one for the program address of the next instruction.

(67) Output Return Jump—This instruction executes a return jump sequence for certain conditions determined by the status of the C-register. The branch condition designator **j** is interpreted in a special way for this instruction to designate which C-register will be examined. If the C-register **j** indicator bit is in the *zero* state, indicating no information in the register, then the return jump condition is satisfied.

If the selected indicator bit is in the *one* state, then the return jump condition is not satisfied. If the return jump condition is not satisfied, the next sequential instruction in the current sequence is executed in a normal manner. If the return jump condition is satisfied, the following sequence is performed:

Clear the X-register and transmit (P) to the lower order 15 bits of X. Clear the P-register and transmit the lower order 15 bits of (Y) to P. Next, store the lower order 15 bits of (X) in the lower order 15 bits of storage address (P)—leaving the higher order 15 bits undisturbed. Then increase (P) by one for the program address of the next instruction.

(70) Initiate Repeat—Initiate a repeat mode of operation which will cause the next sequential instruction to be repeated (Y) times, or until a branch condition is satisfied, whichever occurs first. Maintain the number of remaining executions in B-register 7. Only the lower order 15 bits of (Y) are used in this instruction. The branch condition designator **j** is interpreted in a special way for this instruction. The mode of modification of the repeated instruction is specified by the **j** designator as

j = 0 or 4	Do not modify the repeated instruction after each individual execution.

j = 1 or 5	Increase Y by one after each execution of the repeated instruction.
j = 2 or 6	Decrease Y by one after each execution of the repeated instruction.
j = 3 or 7	Repeat the initial B-register modification of the repeated instruction before each execution.

This instruction is implemented by the following sequence: Clear B-register 7. Transmit the lower order 15 bits of (Y) to B-register 7. Record the lower two bits of the branch condition designator **j** for interpretation during the repeat mode. Establish the repeat mode of operation, and read the next sequential instruction.

(71) Index Skip—If the contents of B-register **j** are equal to (Y), skip the next instruction in the current sequence, proceed to the following instruction, and clear B-register **j**.

If the contents of B-register **j** are not equal to (Y), proceed to the next instruction in the sequence in a normal manner. Increase the contents of B-register **j** by one.

The branch condition designator **j** is used to designate the selected B-register in this instruction and is not available for its normal function. Only the lower order 15 bits of (Y) are used in the comparison with B-register **j**.

(72) Index Jump—If the content of B-register **j** is non zero, execute a jump in the program address to address (Y). Reduce the contents of B-register **j** by one.

If the contents of B-register **j** are zero, proceed to the next instruction in a normal manner. Do not alter the contents of B-register **j**.

The branch condition designator **j** is used to designate the selected B-register in this instruction and is not available for its normal function. If the jump condition is satisfied, then the lower order 15 bits of (Y) become the address of the next instruction, and the beginning of a new program sequence. The higher order 15 bits of (Y) are not used in this instruction.

(73) Initiate Input Transfer—This instruction establishes an input block transfer via C-register **j** to magnetic core storage with an initial storage address (Y). Subsequent to this instruction, the individual transfers will be executed at a rate usually determined by external devices. The storage address initially established by this instruction will be advanced by one, following each individual transfer. The current next address will be maintained throughout the transfer in B-register 6. This mode will continue until it is superseded by a subsequent initiation of an input or output transfer.

This instruction is implemented as follows: Set the C-register status designator for the last previously assigned transfer mode to input and output inactive and clear B-register 6. Transmit the lower order 15 bits of (Y) to B-register 6. Record the **j** designator value for reference during the transfer. Set the C-register **j** status designator to input mode active. Proceed to the next instruction.

The branch condition designator **j** is used to specify the selected C-register for this instruction, and is not available for its normal function.

(74) Initiate Output Transfer—This instruction establishes an output block transfer via C-register **j** from magnetic core storage, with an initial storage address (Y). Subsequent to this instruction, the individual transfers will be executed at a rate usually determined by external devices. The storage address initially established by this instruction will be advanced by one following each individual transfer. The current next address will be maintained throughout the transfer in B-register 6. This mode will continue until it is superseded by a subsequent initiation of an input or output transfer.

This instruction is implemented as follows: Set the C-register status designator for the last previously assigned transfer mode to input and output inactive and clear B-register 6. Transmit the lower order 15 bits of (Y) to B-register 6. Record the **j** designator value for reference during the transfer. Set the C-register **j** status designator to output mode active. Proceed to the next instruction.

The branch condition designator **j** is used to specify the selected C-register for this instruction, and is not available for its normal function.

(75) Initiate Input Buffer—This instruction establishes an input buffer via C-register **j** to magnetic core storage with an initial storage address (Y). Subsequent to this instruction, the individual transfers will be executed at a rate determined by an external device. The storage address initially established by this instruction will be advanced by one, following each individual transfer. The current next address will be maintained throughout the buffer process in the lower order 15 bits of magnetic core storage address **j**. This mode will continue until it is superseded by a subsequent initiation of an input or output buffer via the same C-register—or until the higher order half and the lower order half of storage address **j** contain equal quantities, whichever occurs first.

This instruction is implemented as follows: If $\mathbf{k}=3$, store (Y) in storage location **j**. If $\mathbf{k}\neq 3$, store the lower order 15 bits of (Y) in the lower order half of storage location **j**—leaving the higher order half undisturbed. In either case, set the C-register **j** status designator to input mode active. Proceed to the next instruction.

The branch condition designator **j** is used to specify the selected C-register for this instruction, and is not available for its normal function.

(76) Initiate Output Buffer—This instruction establishes an output buffer via C-register **j** from initial storage address (Y) in magnetic core storage. Subsequent to this instruction, the individual transfers will be executed at a rate usually determined by external devices. The storage address initially established by this instruction will be advanced by one, following each individual transfer. The current next address will be maintained throughout the buffer process in the lower order 15 bits of magnetic core storage address **j**. This mode will continue until it is superseded by a subsequent initiation of an input or output buffer via the same C-register—or until the higher order half and the lower order half of storage address **j** contain equal quantities, whichever occurs first.

This instruction is implemented as follows: If $\mathbf{k}=3$, store (Y) in storage location **j**. If $\mathbf{k}\neq 3$, store the lower order 15 bits of (Y) in the lower order half of storage location **j**—leaving the higher order half undisturbed. In either case, set the C-register **j** status designator to output mode active. Proceed to the next instruction.

The branch condition designator **j** is used to specify the selected C-register for this instruction, and is not available for its normal function.

CONTROL SEQUENCES

The execution of an instruction is divided into a number of lesser operations called sequences. Each performs a logical portion of an instruction. Sequences are further subdivided into steps, each of which performs an elementary manipulation on bits of data, or makes an elementary decision on the basis of a bit of data. A sequence is the smallest portion of a computer instruction which may be executed as an entity. A manual selection on the operation control panel permits an operator manually to step through an instruction one sequence at a time.

Each of the seven sequences provided in the Univac M-460 is identified by a letter. The first four sequences to be described form the execution of a normal instruction. The last three sequences execute a buffer operation on request of external synchronizing signals.

"A" Sequence—Read Next Instruction—This sequence reads the next instruction from storage and modifies the operand address as directed by the address modification designator **b**. The address for the next instruction always resides in the P-register at the beginning of the "A" sequence. As soon as the storage reference has been initiated, (P) is increased by one in anticipation of the next sequential instruction in the current program. The content of a B-register, as specified by the address modification designator **b** is then added to the operand address designator **y**. This addition is accomplished by transferring the contents of the selected B-register to the R-register, and adding (R) to **y**. If **b** = 0, the cleared R-register content is added to the **y** designator.

The "A" sequence for an instruction execution in the repeat mode represents a significant departure from the normal operation described above. During a repeat mode, the previous instruction remains in the U-register and no storage reference is made. An increment is entered in the R-register, as directed by the repeat status designator; and (R) is added to the **y** designator as in a normal execution. The contents of B-register 7 are decreased by one in the repeat mode. If the resultant value of (B7) is *zero*, the repeat mode is terminated. The content of the P-register is not altered in a repeat mode.

"B" Sequence—READ Operand—This sequence reads the operand from storage and performs any preliminary arithmetic appropriate for the particular instruction being implemented. The operand interpretation designator **k** plays a major role in the execution of this sequence. If **k** = 0, 4, or 7, the operand is not obtained from storage but is transferred from the U or A registers to the X-register. If a storage reference is made, the operand is transferred from the Z-register to the X-register—as described in the characteristics of the **k** designator. If arithmetic operations are required prior to procuring the operand, these operations are performed before the operand is transferred to the X-register.

"C" Sequence—Arithmetic Manipulation—This sequence completes all arithmetic processes required for the execution of the instruction. Add, subtract, multiply, divide, and shift operations are all performed in this sequence. No storage references are made. If a jump in the program address is required, the contents of the P-register and the X-register are interchanged during this sequence. If a store operation is required, this sequence is followed by the "D" sequence. If no *store* operation is required, this sequence is followed by the "A" sequence.

"D" Sequence—STORE Operand—This sequence is executed only on those instructions which require a result to be written into storage. In the case of a return jump instruction, the storage location comes from the P-register. In the case of all other instructions, the storage address comes from the U-register. In any case, the quantity contained in the X-register at the beginning of the "D" sequence is stored at the specified storage location, with the **k** designator controlling the transfer from the X-register to the Z-register.

"E" Sequence—READ Buffer Address—This sequence is enabled by an external request for the transfer of a word of data between the storage section and the input-output section of the computer. This request may occur at any time with respect to the main program sequences. Hence, the "E" sequence may be executed between any two of the normal sequences. Once this sequence has been initiated, the normal sequences are delayed until the entire buffer operation has been completed. If a buffer operation is required, this sequence reads an address from a special storage location, and enters it in the R-register. This sequence then enables the "F" sequence. If a transfer operation is required, the "E" sequence reads an address from B-register 6, performs the transfer between the required C-register and storage, and then increases the address by one and returns it to B-register 6. In this case no other buffer sequence is required.

"F" Sequence—Transfer Buffer Word—This sequence buffers a word of input or output data between a selected C-register and the storage address contained in the R-register. The "G" sequence is then enabled.

"G" Sequence—STORE Buffer Address—This sequence increases the buffer address contained in the R-register by one, and stores the result in the special storage location from which it originally came. During this storage reference, the new buffer address being recorded is compared with a terminal address contained in the upper half of the same storage location. If the two addresses are equal, the buffer mode is terminated. In either case, the buffer operation for this word of input or output data is completed; and the main program sequences resume.

SPECIAL MODES OF OPERATION

The M-460 Computer performs a number of complex operations which may be identified as special modes of operation. Several of these special modes require the existence of magnetic core storage locations as implicit addresses for control information associated with the functioning of the special mode. These special storage locations are assigned to beginning values of the address range as follows:

00000—initial starting address from a master clear
00001—buffer control word for C-register 1
00002—buffer control word for C-register 2
00003—buffer control word for C-register 3
00004—buffer control word for C-register 4
00005—buffer control word for C-register 5
00006—buffer control word for C-register 6
00007—buffer control word for C-register 7
00010—real time clock
00011—interrupt instruction
00012—interrupt exit
00013—interrupt entrance

Repeat Mode of Operation—A repeat mode of operation is initiated by the execution of an initiate repeat (70) instruction. The purpose of the repeat mode is to perform multiple executions of an instruction with minor variations in the address of the operand. This instruction is used to process a list of data words with a minimum of storage references for control purposes. Such operations as searching a list of words for coincidence with a reference word, or adding a column of numbers, may be performed much more rapidly with this mode than with an equivalent subroutine. The branch condition designator j plays a major role in this mode. It permits the repeat mode to be terminated when a particular execution satisfies the branch condition. For example, a search for coincidence with a reference word could utilize the branch condition designator in the repeated instruction. When and if the coincident item is located, the repeat mode is terminated. The address of the selected word can then be obtained by subtracting the number of incompleted executions, as indicated in B-register 7, from the terminal address of the search.

The repeat mode is implemented through a control designator which modifies the execution of the "A" sequence when the repeat mode is active. A nonzero value of this designator suppresses the procurement of the next sequential instruction from storage. The operand address of the repeated instruction is modified during the "A" sequence of each execution of the instruction to advance the address of the next item in the list being processed. The number of incompleted executions is maintained in B-register 7 throughout the repeat mode.

If a repeat mode is terminated by a branch condition, then the instruction in the storage location following the repeated instruction is skipped. If the repeat mode is terminated by exhausting the repeat count in B-register 7, then the instruction in the storage location following the repeated instruction is executed.

Buffer Mode of Operation—A buffer mode of operation is initiated by the execution of an initiate input buffer (75) or initiate output buffer (76) instruction. The purpose of the buffer mode is to provide input and output communication between the magnetic core storage and external equipment via the communication registers. This operation is independent of, and in parallel with, the main computer program. Provisions are made in the Univac M-460 for up to seven such independent buffering operations at one time—one through each of the seven communication registers.

Each buffer control utilizes a special magnetic core storage location for control information. Two 15-bit addresses are retained in this storage location, and are referenced and modified as the buffer operation is performed. The higher order 15 bits of a buffer control word represent the terminal address plus one for the buffered data. Upon initiation of the buffer, the lower order 15 bits of a control word represent the initial storage address of the buffered data. As each item is transferred, the lower portion of the buffer control word is increased by one until this portion is identical with the upper portion of the control word. A control signal is generated by the equality of the upper and lower halves of a control word which terminates the buffer mode for the C-register involved.

All data transferred into or out of the magnetic core storage via a buffer mode are treated as 30-bit words. If the C-register involved in the buffer mode provides less than 30 bits, then the higher order bits are always interpreted as *zero*. The initial buffer address is included in, the terminal address excluded from, the storage addresses used in a buffer mode.

The individual steps in a buffer mode of operation are initiated by the entry, or removal, of a data word in the associated C-register. The main program sequences are suspended by these events, and a three-sequence operation is performed to complete the buffer process.

- The "E" sequence reads the control word associated with the active C-register, and enters the lower order 15 bits into the R-register.
- The "F" sequence uses the address in the R-register to transfer the data word in the C-register to or from the magnetic core storage.
- The "G" sequence increases the address in the R-register by one, and records the result in the lower order 15 bits of the special storage location. If the upper and lower halves of this storage location are equal, the buffer mode is terminated.

If several requests for buffer action occur at one time, they are processed in the order of the C-register number, with C-register 7 having the top priority. A designator holds the selected C-register number throughout the three buffer sequences. As soon as one request has been satisfied, and the designator has been cleared, a second request may become active and the corresponding C-register number entered in the designator. The main program sequences resume when the designator remains cleared on completion of a buffer sequence.

Transfer Mode of Operation—A transfer mode of operation is initiated by the execution of an initiate input transfer (73) or initiate output transfer (74) instruction. The purpose of the transfer mode is to move rapidly a block of data into or out of the magnetic core storage. This mode is similar to the buffer mode of operation. However, it differs in that the transfer mode does not terminate at a predetermined address. The transfer mode must be terminated by a main program instruction—which reassigns the mode to another C-register or disables the mode by assigning it to C-register 0. The transfer mode also differs from the buffer mode in that the time required to complete the handling of each word is considerably less than that for the buffer mode. Only one C-register can be in the transfer mode at one time.

The principle application of the transfer mode of operation is a block transfer of data from one location in magnetic core storage to another location in the same storage. This is accomplished by establishing a transfer mode via an arbitrary C-register, and then referencing the C-register with a repeated main program instruction.

All data transferred into or out of the magnetic core storage via a transfer mode are treated as 30-bit words. If the C-register involved in the transfer mode provides less than 30 bits, then the higher order bits are always interpreted as *zero*. The individual steps in a buffer mode of operation are initiated by the entry, or removal, of a data word in the associated C-register. The main program sequences are suspended by these events, and a special "E" sequence is performed to complete the transfer process. This sequence transfers the word between the active C-register and the storage address specified by B-register 6. The contents of B-register 6 are then increased by one.

The C-register, which is active in a transfer mode, is always recorded in a control designator. If a buffer and a transfer mode are assigned to the same C-register, the transfer mode is effective and the buffer mode is ignored. If a transfer mode is assigned to C-register 0, the mode becomes inactive.

Advance Clock Mode of Operation—Magnetic core storage address 00010 functions as a 30-bit counter, advancing one count each millisecond. This provision is made so that real-time measurements may be included in the computation. The advancing of this counter is accomplished by a buffer type operation under the control of a one kilocycle crystal oscillator. Once each millisecond, a control designator is set to indicate the need for advancing the clock. The main program "E" and "G" sequences are executed as if a buffer operation were being performed. These sequences advance the lower order 15 bits of (00010). If at the end of the "G" sequence $(R) = 0$, a special "F" sequence is enabled, which advances the higher order 15 bits of (00010) by one.

Interrupt Mode of Operation—The interrupt mode of operation permits an M-460 program to be interrupted by an unexpected request signal from external equipment. Upon interruption by such a signal, the computer executes the instruction stored at a special storage location (00011). If this instruction were not a jump instruction, the program would then continue as if it had not been interrupted. A more usual instruction stored at address 00011 would be a return jump to address 00012. This would record the current program address in the lower order 15 bits of (00012), and commence a routine beginning at address 00013. This routine would evaluate the reason for the interruption via

C-register communication with external equipment. When completed, the routine would execute a jump instruction at address 00012 and return to the main program.

An interruption of a program and the resulting execution of the instruction at address 00011 cause the interrupt designator to be set. This disables the interrupt request line from the external equipment, and prevents a second interruption from superseding the first. Upon completion of the interrupt routine, an instruction is normally read from address 00012 to return to the main program. The process of reading an instruction, not an operand, from address 00012 causes the interrupt request line to be enabled for further interruptions.

COMMUNICATION CABLES

The M-460 Computer communicates with other equipment via a number of communication lines. These communication lines are physically grouped into cables containing 20 twisted pairs of wires and terminating in 37 pin AN connectors. There are a total of 24 of these cables, functionally grouped into three categories. These categories are described below as input, output, and function cables. Each wire in a cable has a function which is identical to a similar wire in any other cable of the same category. Hence, any external equipment can be connected to any C-register as long as the cable categories are observed. Communication registers with fewer than 30 bits transmit and receive zeros in the higher order digits. A potential of -10V is interpreted as a *zero* signal. A potential of 0 volts is interpreted as a *one* signal.

Input Cables (12)—There are a total of 12 input cables provided in the Univac M-460 Computer. Two cables are associated with each of the communication registers, except C-register 3, which is the function register. Each cable contains 30 information lines and 2 control lines. The information lines are twisted with one another, and the control lines are twisted with ground wires. One control line, identified as the *ready line*, signals the external equipment that the C-register is ready for an input. The other control line, identified as the *resume line*, signals the computer that the input information lines are to be sampled and the results entered in the C-register.

Output Cables (6)—There are provided a total of six output cables. One output cable is associated with each of the communication registers, except for C-register 3. Each cable contains 30 information lines and two control lines. The information lines are twisted with one another, and the control lines are twisted with ground wires. One control line, identified as the *ready line*, signals the external equipment that the C-register has been filled and that the output information lines are ready to be sampled. The other control line, identified as the *resume line*, signals the computer that the information has been sampled and that it is no longer needed.

Function Cables (6)—There are a total of six function cables provided in the M-460. Each cable contains outputs from C-register 3 and inputs to C-register 3. The cables are numbered 1 through 6, corresponding to values of the output channel designator and the input channel designator which activate them. Function channels 1 and 2 have fifteen active input lines and twelve active output lines. Function channels 3, 4, 5, and 6 have six active input lines and six active output lines. Information lines are twisted with one another, and the control lines are twisted with ground wires.

Each cable contains 15 input information lines, 12 output information lines, and 2 control lines. One control line, identified as the *ready line*, signals the external equipment that C-register 3 has been filled and that the output lines are ready to be translated. The other control line, identified as the *interrupt request line*, may be activated at any time by external equipment to interrupt the computer program. Inputs to C-register 3 are sampled at the time the input channel selection is made.