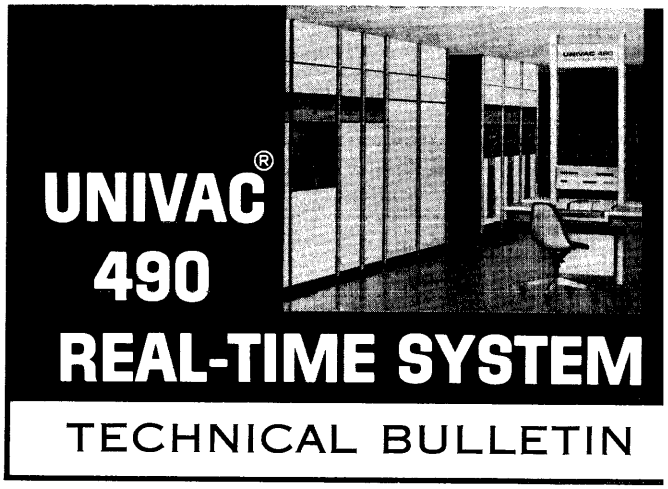

UNIVAC 490
Real-Time Computer

October, 1961

CONTENTS

1. REAL-TIME COMPUTER 1-1

Storage Section 1-1

Octal Notation 1-1

Control Section 1-2

Arithmetic Section 1-2

Input-Output Section 1-2

Registers 1-2

Operational Registers 1-3

Transient Registers 1-3

Operator Console 1-4

Maintenance Panel 1-4

Wired Memory 1-5

2. INSTRUCTIONS 2-1

Notation 2-1

Instruction Word 2-2

f Designator 2-2

y Designator 2-2

j Designator 2-2

k Designator 2-2

b Designator 2-2

3. TRANSFER INSTRUCTIONS 3-1

4. ARITHMETIC INSTRUCTIONS 4-1

Subtraction 4-1

Addition 4-2

Multiplication 4-12

Division 4-14

Negative Zero Quotients and Remainders 4-14

5. SHIFT INSTRUCTIONS 5-1

6. LOGICAL INSTRUCTIONS 6-1

Logical Product 6-1
Selective Set 6-1
Selective Clear 6-2
Selective Complement 6-2
Selective Substitute 6-2

7. MODIFYING INSTRUCTIONS 7-1

8. JUMP INSTRUCTIONS 8-1

9. INPUT-OUTPUT INSTRUCTIONS 9-1

Input-Output Instruction Word 9-1
f Designator 9-1
y Designator 9-1
$\hat{j}$ Designator 9-1
$\hat{k}$ Designator 9-1
$\hat{j}\hat{k}$ Combinations 9-1
b Designator 9-2
Input-Output Buffers 9-2
Function Words 9-6
Internal Interrupt 9-7
External Interrupt 9-7
Input-Output Priority Structure 9-8

1. REAL-TIME COMPUTER

The heart of the UNIVAC®490 Real-Time System is a stored program binary computer designed to process large quantities of data in both batch-processing and real-time modes. The Computer provides large internal magnetic core storage, programming flexibility, a versatile input-output section, and solid-state arithmetic and logical circuitry that performs tens of thousands of processing operations every second.

Other important features of the Real-Time Computer are:

- Access time to all storage locations of 1.9 microseconds; ability to store and to select information randomly.
- 30–bit word length with a 15–bit half-word option.
- Repertoire of 62 basic instructions which can be modified to produce over 25,000 different instructions.
- Single address instructions with provision for address modification.
- Multiple program capabilities.
- Ability to perform rapid data exchanges with external equipment without main program attention.
- Real-time clock for automatically initiating various Computer operations at predetermined times.
- Parallel one's complement binary notation.

STORAGE SECTION

The internal storage in the Computer consists of thousands of ferrite cores that are mounted within printed-circuit frames. Each core is capable of assuming either of two stable magnetic states; one state represents binary 0; the other, binary 1.

At the option of the user, the Computer is available with a storage capacity of either 16,384 or 32,768 computer words. Since information in storage is randomly selected, access to all addresses is the same; that is, words can be inserted into or removed from any address in storage at a rate of six microseconds per word. The basic internal data word is shown in Figure 1–1.

As shown in the diagram, bit position 29 contains the sign. If this bit position contains a binary 0, the quantity represented is positive. If it contains a binary 1, the quantity is negative.

Octal Notation

The UNIVAC 490 Real-Time Computer is a binary computer; consequently, all computer words appear as combinations of binary 1's and 0's. Since a computer word consists of 30 bits, this tends to be cumbersome. For this reason, binary notation is expressed in octal form. The conversion from binary to octal notation simply involves dividing the binary digits into consecutive sets of three from right to left, and then reading these sets in decimal. For example, a full core storage system requires the use of 32,768 storage addresses. Representation of the upper limit storage in binary notation requires the use of 15 bits. This same decimal number, however, can be represented by five octal digits.

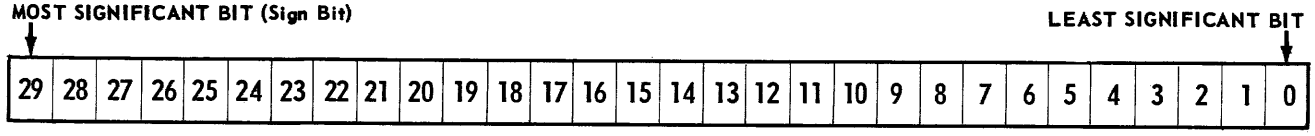

Figure 1-1. Basic Internal Data Word

DECIMAL	32,767 (UPPER LIMIT ADDRESS)
BINARY	111 111 111 111 111
OCTAL	7 7 7 7 7

It should be noted that the working digits in the octal system are 0 through 7. The word "octal" means "eight;" therefore, when counting in octal notation, the number after 7 is 10. For ease of programming, all quantities, function code values, instruction designator values, the operand address, and the operand itself are expressed in octal notation.

CONTROL SECTION

In addition to the storage section, the Computer has three other sections, control, arithmetic, and input-output (Figure 1–2). The control section governs the operations that take place during the sequential execution of the instructions. It also coordinates the flow of data between the arithmetic and storage sections.

ARITHMETIC SECTION

The arithmetic section is composed of the circuits and registers that are used to perform arithmetic and logical operations.

INPUT-OUTPUT SECTION

The input-output section is composed of the circuits, input gates, and output registers that are used to transfer data to or from the Computer.

REGISTERS

The UNIVAC 490 Real-Time Computer contains a number of registers which hold information during computation. These registers, designated by a letter or letter-numeral combination, are interconnected by parallel transmission paths. During processing, information flows to and from the registers via these paths.

The registers fall into two categories: operational and transient. Operational registers are referred to in the functional description of an instruction. Information that is placed in these registers is retained until it is replaced by new information. Transient registers are temporary storage locations that are used in the manipulation of instruction and data words during the execution of an instruction. These registers are not referenced by the instruction and do not retain information from one operation to the next.

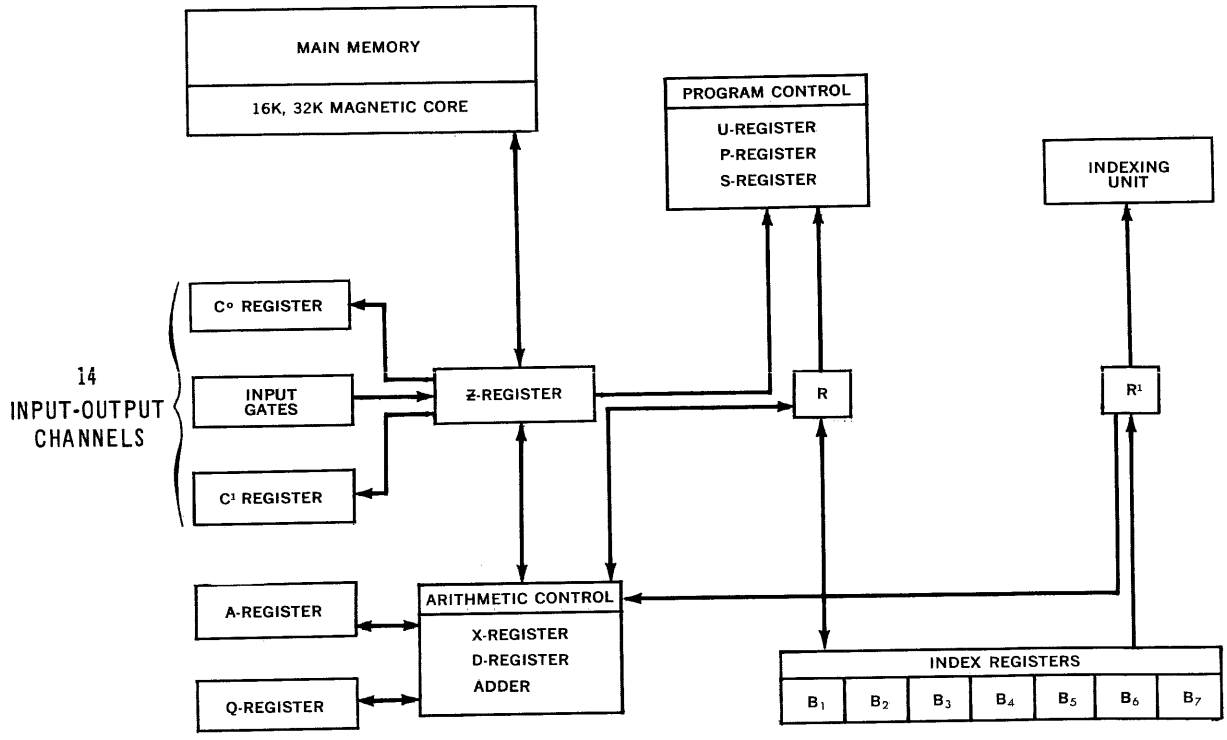

Figure 1-2. Simplified Logic Diagram of the UNIVAC 490 Real-Time Computer

Operational Registers

A-Register

The *A*-register or accumulator is the principal 30–bit arithmetic register. It is primarily used for arithmetic and shifting operations. In arithmetic operations, the result is usually retained in the *A*-register for use in later program steps. For example, after addition or subtraction, the sum or difference is retained in the accumulator; after multiplication, the most significant half of the product is retained in the accumulator; after division, the remainder is retained in the accumulator.

In shifting operations, the contents of the *A*-register may be shifted right or left. Left shifts are circular. In right shifts the sign bit is extended by the number of bit positions shifted and the bits shifted from the lower-end are lost.

Q-Register

The *Q*-register is a 30–bit auxiliary arithmetic register that has adding, shifting, and logical properties. Its principal function is to assist the *A*-register in multiply, divide, and logical operations.

In the case of shifting operations, the contents of the *Q*-register may be shifted right or left in the same manner as the *A*-register.

A- and Q-registers in combination

Multiply, divide, and certain shift instructions utilize the *A*- and *Q*-registers as a single 60–bit register. To illustrate, the *Q*-register holds the multiplier at the beginning of a multiply operation and at the end the double-length product is in the *AQ*-register, that is, the upper half of the product is in the *A*-register and the lower half is in the *Q*-register. In a divide operation the *AQ*-register holds the dividend and at the end the quotient is in the *Q*-register and the remainder is in the *A*-register.

In shifting operations, the 60–bit contents of the *AQ*-register (the upper half of which is the *A*-register) may be shifted right or left in the same manner as either the *A*-register or the *Q*-register. When the *AQ*-register is shifted to the right, bits from the lower-end of the *A*-register are shifted into the upper-end of the *Q*-register and the bits that are shifted from the lower-end of the *Q*-register are lost. In a left shift, bits from the upper-end of the *Q*-register are shifted into the lower-end of the *A*-register and the bits that are shifted from the upper-end of the *A*-register fill in on the lower-end of the *Q*-register.

P-Register

The *P*-register or program address counter is a 15–bit register that 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 1. When Jump instructions are executed, the *P*-register is cleared and a new program address is entered.

B-Registers (Index Registers)

The *B*-registers, numbered *B*1 through *B*7, are 15–bit registers whose contents are used to increment the operand address before execution of an instruction may also be used to index program loops. In addition, the *B*7-register is used as a counter in the repeat mode where a selected instruction is executed the number of times specified (covered later in the discussion of the 70 instruction).

Transient Registers

X-Register

The *X*-register, a 30–bit register with complementing properties, is used as an arithmetic communication register. The *X*-register receives the operand from storage during all arithmetic operations. All communication between the *A*- and *Q*-registers and the other operational registers or the adder output is via the *X*-register.

S-Register

The *S*-register is a 15–bit register which holds the storage address during storage references. At the beginning of the 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.

Z-Register

The *Z*-register is a 30–bit register that serves as an operand buffer for storage references. Dur- 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 the *Z*-register corresponding to the bits in storage. During the write portion of the storage access period, the *Z*-register controls the inhibit circuits in order to write or restore the disturbed storage address. Input data from the input channels is gated directly to the *Z*-register.

K-Register

The *K*-register is a 6–bit register that functions as a shift counter for shift operations and all arithmetic operations involving shifts. The maximum shift count permitted is 59_{10}. Multiply and divide operations are controlled by presetting the *K*-register to 30_{10}. The *K*-register then counts the operational steps.

U-Register

The *U*-register or program control register is a 30–bit register that holds the instruction word during the execution of an operation. The function code and the various instruction designators are translated from the appropriate sections of this register. If an address modification is required before execution, the contents of the appropriate *B*-register are added to the low order 15 bits of the *U*-register.

R-Register

The *R*-register is a 15–bit register that functions as a communications register for all internal transmissions to the *B*-registers.

R'-Register

The *R'*-register is a 15–bit register that functions as a communications register for all internal transmissions from the *B*-registers. During address modification this register holds the incrementing quantity.

C-Registers

The *C*-registers are 30–bit communication buffer registers through which computer data is synchronized. There are two *C*-registers, *C*0 and *C*1. The *C*0-register is used to communicate output data to peripheral devices on 12 different channels. The *C*1-register is used to communicate output data on two different channels to other computers. Input data is gated directly to the *Z*-register.

OPERATOR'S CONSOLE

The Operator's Console is used for monitoring internal operations. It consists of an alphanumeric keyboard, a printer, and an indicator panel. The keyboard is used to enter data or changes to the program and the printer provides type-outs of program generated information. The indicator panel contains indicators that inform the operator of the status of the program in progress and switches that allow various manual operations to be performed.

MAINTENANCE PANEL

The maintenance panel is used for debugging and preventive maintenance operations. To aid in these operations the maintenance panel is provided with a series of indicators and switches. Included among the indicators are the following registers:

1. P-register
2. C0-register
3. C1-register
4. Q-register
5. A-register
6. B1-register
7. B2-register
8. B3-register
9. B4-register
10. B5-register
11. B6-register
12. B7-register
13. U-register
14. S-register
15. X-register
16. Z-register
17. R-register
18. R'-register

In addition to the above indicators, switches are provided that allow:

- The execution of consecutive program steps at a low rate.
- The execution of one consecutive Computer clock phase (¼ of a cycle) for each depression of a switch.
- The execution of one consecutive program step for each depression of a switch.
- Operation that is normal except that the Computer does not stop when it executes a programmed stop instruction.
- The Real-Time Clock to be disconnected.
- The Increment Clock to be disconnected.
- The automatic recovery feature to be disconnected.

- The functions of the following switches on the maintenance panel to be transferred to the Operators's Console: Jump switches 1, 2, and 3; the Bootstrap switch (maintenance panel) to the Start switch (Operator's Console); Instruction Step switch (maintenance panel) to Stop switch (Operator's console).

WIRED MEMORY

A permanent memory is built into the Computer to provide for automatic reading in of new programs and automatic error recovery. It consists of sixteen 30–bit words of storage and is wired to fit the specialized needs of the Computer user. The wired memory may be accessed by a program, but can only be changed manually (by maintenance personnel). The addresses of the words in wired memory parallel storage addresses 00000 through 00017 in the Computer. Whether the Computer utilizes the words in wired memory or those in core storage, depends upon the position of the Wired Memory switch (Bootstrap switch) on the maintenance panel. The positions of this switch are:

- Automatic Recovery
- Neutral
- Bootstrap

When this spring-loaded switch is in the Bootstrap position, the Computer executes the program starting at address 00000 in the wired memory.

When the switch is in the Automatic Recovery position, and a fault interrupt occurs, the Computer executes the program starting at wired memory address 00014. A fault interrupt is caused when the Incremental Clock interrupts on overflow, an illegal function code (00 or 77) occurs, or the Incremental Clock is not updated.

When the switch is in the Neutral position, the Computer ignores the wired memory, and any references to addresses 00000 through 00017 apply to these addresses in core storage.

2. INSTRUCTIONS

The UNIVAC 490 Real-Time Computer has a repertoire of 62 basic instructions that can be modified to produce over 25,000 different instructions. These instructions fall into seven categories: *transfer, arithmetic, shift, logical, modifying, jump* and *input-output*. Each category is covered in a separate chapter.

NOTATION

In the succeeding discussions, the following conventions are used:

Y Operand.

$\bar{y}$ The low-order 15 bits contained in the instruction word after *B*-register modification.

Y_L The low-order 15 bits contained in the storage location at address $\bar{y}$.

Y_U The high-order 15 bits contained in the storage location at address $\bar{y}$.

$X\bar{y}$ A 30-bit operand whose lower half is $\bar{y}$, the low-order 15 bits contained in the instruction word after *B*-register modification; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 of $\bar{y}$ is a, 0 or it will be all 1's if bit 14 is a 1.

XY_L A 30-bit operand whose lower half is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is an extension of the sign. The upper half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1.

XY_U A 30-bit operand whose lower half is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is an extension of the sign. The upper half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1.

A The *A*-register (30 bits).

Q The *Q*-register (30 bits).

AQ The *A*-register combined with *Q*-register to form a 60-bit register. The *A*-register is the most significant half of this register.

B_j The particular *B*-register specified by *j* (15 bits).

P The program address register (15 bits).

R The remainder.

( ) The contents of the register or storage location enclosed within the parentheses.

( )' The complement of the contents of the register or storage location enclosed within the parentheses.

( )i The initial contents of the register enclosed within the parentheses.

( )f The final contents of the register enclosed within the parentheses.

( )n The nth bit position of the register enclosed within the parentheses.

NI The next instruction.

INSTRUCTION WORD

The instruction word format is shown in Figure 2-1.

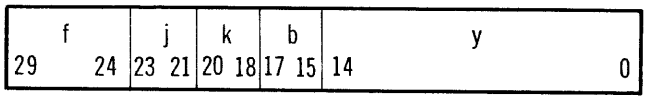

Figure 2-1. Instruction Word

f DESIGNATOR

The function code designator, *f*, is a 6-bit code that specifies the operation to be performed.

y DESIGNATOR

The operator designator, *y*, is a 15-bit code that represents either the operand or the operand address before *B*-register modification.

j DESIGNATOR

The branch-condition designator, *j*, is a 3-bit code that may be interpreted as a skip or jump-condition designator, a register designator, or a repeat modification designator.

k DESIGNATOR

The operand-interpretation designator, *k*, is a 3-bit code that controls where the operand is procured from and/or where it is stored.

b DESIGNATOR

The operand address modification designator, *b*, is a 3-bit code that specifies the *B*-register that contains the quantity that is added to *y* to form the operand or operand address.

B-Register (Index Register) Modification Of Instructions

B-register modification consists of adding the contents of the *B*-register specified by the *b* designator in the instruction word to *y*, the low-order 15 bits of this word, before any storage reference is made. This process takes place in the program control register, *U*; consequently, the instruction word as it appears in storage, is not altered. The sum formed by this addition is $\bar{y}$.

One of eight *B*-registers, numbered 0 through 7, may be specified by the *b* designator in the instruction word. *B*-register 0 is not a physical register like the other seven. If this register is specified, that is, *b* = 0 in the instruction word, the result is the same as if the specified *B*-register contained all 0's. In this case, as in all others, *B*-register modification consists of forming $\bar{y} = y + (B)j$. This addition is a 15-bit binary addition with end-around carry. An end-around carry results when a carry is generated in the addition of the highest order bits; a binary 1 is added to the lowest order bit in the sum. The following example illustrates this concept:

y = 77775 = 111111111111101	(the low-order 15 bits contained in the instruction word)
$(B)j$ = 00005 = 000000000000101	(the contents of the specified *B*-register)
1	(end-around carry)
$\bar{y}$ = 00003 = 000000000000011	(the low-order 15 bits contained in the instruction word after *B*-register modification)

It should be noted that *B*-register modification cannot be used to generate $\bar{y}$ = 00000 unless both *y* and $(B)j$ = 00000. This limitation is imposed because the nature of end-around carry addition is such that it is mathematically impossible to generate $\bar{y}$ = 00000 except in the case cited above.

3. TRANSFER INSTRUCTIONS

Transfer instructions either transfer data that is contained in a storage location to a register or store the contents of a register in a storage location.

ENTER Q

CLASS: Read

FUNCTION CODE: 10

MNEMONIC: ENT · Q

OPERATION: $Y \longrightarrow Q$

DESCRIPTION: This instruction transfers a 30-bit operand to the Q-register.

k DESIGNATORS: The operand transferred to the Q-register is derived as follows:

k = 0: $\bar{y} \longrightarrow Q$. The lower half of the operand is $\bar{y}$, the low-order 15 bits contained in the instruction word after B-register modification; the upper half is all 0's.

k = 1: $Y_L \longrightarrow Q$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is all 0's.

k = 2: $Y_U \longrightarrow Q$. The lower half of the operand is the high-order 15 bits contained in the storage at address $\bar{y}$; the upper half is all 0's.

k = 3: $Y \longrightarrow Q$. The 30-bit operand is contained in the storage location at address $\bar{y}$.

k = 4: $X\bar{y} \longrightarrow Q$. The lower half of the operand is $\bar{y}$, the low-order 15 bits contained in the instruction word after B-register modification; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 of $\bar{y}$ is a 0, or it will be all 1's if bit 14 is a 1.

k = 5: $XY_L \longrightarrow Q$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is an extension of the sign bit. The upper half of the

operand will be all 0's if bit 14 in the storage location is a 0 or it will be all 1's if bit 14 is a 1.

k = 6: $XY_U \longrightarrow Q$. The lower half of the operand and is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 29 in the storage location is a 0 or it will be all 1's if bit 29 is a 1.

k = 7: $A \longrightarrow Q$. The operand is the 30 bits contained in the A-register.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.
j = 1: skip *NI*
j = 2: skip *NI* if $(Q)_f$ is + or + 0.
j = 3: skip *NI* if $(Q)_f$ is - or - 0.
j = 4: skip *NI* if (A) is + 0.
j = 5: skip *NI* if (A) is not + 0.
j = 6: skip *NI* if (A) is + or + 0.
j = 7: skip *NI* if (A) is - or - 0.

ENTER A

CLASS: Read

FUNCTION CODE: 11

MNEMONIC: ENT · A

OPERATION: $Y \longrightarrow A$

DESCRIPTION: This instruction transfers a 30-bit operand to the A-register.

k DESIGNATORS: The operand transferred to the A-register is derived as follows:

k = 0: $\bar{y} \longrightarrow A$. The lower half of the operand is $\bar{y}$, the low-order 15 bits contained in the instruction word after B-register modification; the upper half is all 0's.

k = 1: $Y_L \longrightarrow A$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is all 0's.

k = 2: $Y_U \longrightarrow A$. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is all 0's.

k = 3: $Y \longrightarrow A$. The operand is the 30-bits contained in the storage location at address $\bar{y}$.

k = 4: $X\bar{y} \longrightarrow A$. The lower half of the operand is y – the low-order 15 bits contained in the instruction word after B-register modification; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 of $\bar{y}$ is a 0 or it will be all 1's if bit 14 is a 1.

k = 5: $XY_L \longrightarrow A$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 in the storage location is a 0 or it will be all 1's if bit 14 is a 1.

k = 6: $XY_U \longrightarrow A$. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 29 in the storage location is a 0 or it will be all 1's if bit 29 is a 1.

k = 7: $(A) \longrightarrow A$. The contents of the A-register remains the same.

j DESIGNATORS: The skip conditions are deter- as follows:

j = 0: no skip.
j = 1: skip *NI*.
j = 2: skip *NI* if (Q) if + or + 0.
j = 3: skip *NI* if (Q) is - or - 0.
j = 4: skip *NI* if $(A)_f$ is + 0.
j = 5: skip *NI* if $(A)_f$ is not + 0.
j = 6: skip *NI* if $(A)_f$ is + or + 0.
j = 7: skip *NI* if $(A)_f$ is - or - 0.

ENTER B_j

CLASS: Read

FUNCTION CODE: 12

MNEMONIC: ENT · B_n

OPERATION: Y→B_j

DESCRIPTION: This instruction transfers a 15-bit operand to a selected B-register. The j designator specifies the selected register; consequently, a skip condition *cannot* be programmed in this instruction.

k DESIGNATORS: The operand transferred to the selected B-register is derived as follows:

k = 0 or 4: $\overline{y} \longrightarrow B_j$. The operand is the low-order 15 bits contained in the instruction word after B-register modification.

k = 1, 3, or 5: $Y_L \longrightarrow B_j$. The operand is the low-order 15 bits contained in the storage location at address $\overline{y}$.

k = 2, or 6: $Y_U \longrightarrow B_j$. The operand is the high-order 15 bits contained in storage location at address $\overline{y}$.

k = 7: $A_L \longrightarrow B_j$. The operand is the low-order 15-bits contained in the A-register.

j DESIGNATORS: The selected B-register is specified as follows:

j = 0: no operation is performed and the program advances to the next instructions.

j = 1: B-register 1.

j = 2: B-register 2.

j = 3: B-register 3.

j = 4: B-register 4.

j = 5: B-register 5.

j = 6: B-register 6.

j = 7: B-register 7.

STORE Q

CLASS: Store

FUNCTION CODE: 14

MNEMONIC: STR · Q

OPERATION: $(Q) \longrightarrow Y$

DESCRIPTION: This instruction stores the contents of the Q-register in a storage location.

k DESIGNATORS: The operand that is stored in the storage location is derived as follows:

k = 0: $(Q)' \longrightarrow Q$. The contents of the Q-register are complemented.

k = 1: $(Q)_L \longrightarrow Y_L$, Y_U is undisturbed. The low-order 15 bits of the Q-register are stored in the lower half of the storage location at address $\overline{y}$; the upper half of the storage location is undisturbed.

k = 2: $(Q)_L \longrightarrow Y_U, Y_L$ is undisturbed. The low-order 15 bits of the Q-register are stored in the upper half of the storage location at address $\overline{y}$; the lower half of the storage location is undisturbed.

k = 3: $(Q) \longrightarrow Y$. The contents of the Q-register are stored in the storage location at address $\overline{y}$.

k = 4: $(Q) \longrightarrow A$. The contents of the Q-register are stored in the A-register.

k = 5: $(Q)'_L \longrightarrow Y_L$, Y_U is undisturbed. The complement of the low-order 15 bits of the Q-register is stored in the lower half of the storage location at address $\overline{y}$; the upper half of the storage location is undisturbed.

k = 6: $(Q)'_L \longrightarrow Y_U$, Y_L is undisturbed. The complement of the low-order 15 bits of the Q-register is stored in the upper half of the storage location at address $\overline{y}$; the lower half of the storage location is undisturbed.

k = 7: $(Q)' \longrightarrow Y$. The complement of the contents of the Q-register is stored in the storage location at address $\overline{y}$.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.
j = 1: skip *NI*.
j = 2: skip *NI* if $(Q)_f$ is + or + 0.
j = 3: skip *NI* if $(Q)_f$ is - or - 0.
j = 4: skip *NI* if $(A)_f$ is + 0
j = 5: skip *NI* if $(A)_f$ is not + 0.
j = 6: skip *NI* if $(A)_f$ is + or + 0.
j = 7: skip *NI* if $(A)_f$ is - or - 0.

STORE A

CLASS:	Store
FUNCTION CODE:	15
MNEMONIC:	STR · *A*
OPERATION:	$(A) \longrightarrow Y$

DESCRIPTION: This instruction stores the contents of the *A*-register in a storage location.

j DESIGNATORS: The operand that is stored in the storage location is derived as follows:

k = 0: $(A) \longrightarrow Q$. The contents of the *A*-register are stored in the *Q*-register.

k = 1: $(A)_L \longrightarrow Y_L$, Y_U is undisturbed. The low-order 15 bits of the *A*-register are stored in the lower half of the storage location at address $\overline{y}$; the upper half of the storage location is undisturbed.

k = 2: $(A)_L \longrightarrow Y_U$, Y_L is undisturbed. The low-order 15 bits of the *A*-register are stored in the upper half of the storage location at address $\overline{y}$; the lower half of the storage location is undisturbed.

k = 3: $(A) \longrightarrow Y$. The contents of the *A*-register are stored in the storage location at address $\overline{y}$.

k = 4: $(A)' \longrightarrow A$. The contents of the *A*-register are complemented.

k = 5: $(A)'_L \longrightarrow Y_L$, Y_U is undisturbed. The complement of the low-order 15-bits of the *A*-register is stored in the lower half of the storage location at address $\overline{y}$; the upper half of the storage location is undisturbed.

k = 6: $(A)'_L \longrightarrow Y_U$, Y_L is undisturbed. The complement of the low-order 15-bits of the *A*-register is stored in the upper half of the storage location at address $\overline{y}$; the lower half of the storage location is undisturbed.

k = 7: $(A)' \longrightarrow Y$. The complement of the contents of the *A*-register is stored in the storage location at address $\overline{y}$.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.
j = 1: skip *NI*.
j = 2: skip *NI* if $(Q)_f$ is + or + 0.
j = 3: skip *NI* if $(Q)_f$ is - or - 0.
j = 4: skip *NI* if $(A)_f$ is + 0.
j = 5: skip *NI* if $(A)_f$ is not + 0.
j = 6: skip *NI* if $(A)_f$ is + or + 0.
j = 7: skip *NI* if $(A)_f$ is - or - 0.

STORE B_j

CLASS:	Store
FUNCTION CODE:	16
MNEMONIC:	STR · B_n
OPERATION:	$(B)_j \longrightarrow Y$

DESCRIPTION: This register stores the contents of a selected *B*-register in a storage location The *j* designator is used to specify the selected register; consequently, a skip condition *cannot* be be programmed in this instruction.

k DESIGNATORS: The operand that is stored in the storage location is derived as follows:

k = 0: $(B)_j \longrightarrow Q_L$, 0's $\longrightarrow Q_U$. The contents of the *B*-register are stored in the lower half of the *Q*-register; the upper half of the *Q*-register is all 0's.

k = 1: $(B)_j \longrightarrow Y_L, Y_U$ is undisturbed. The contents of the *B*-register are stored in the lower half of the storage location at address $\bar{y}$; the upper half of the storage location is undisturbed.

k = 2: $(B)_j \longrightarrow Y_U, Y_L$ is undisturbed. The contents of the *B*-register are stored in the upper half of the storage location at address $\bar{y}$; the lower half of the storage location is undisturbed.

k = 3: $(B)_j \longrightarrow Y_L$, 0's $\longrightarrow Y_U$. The contents of the *B*-register are stored in the lower half of storage location at address $\bar{y}$; the upper half of the storage location is all 0's.

k = 4: $(B)_j \longrightarrow A_L$, 0's $\longrightarrow A_U$. The contents of the *B*-register are stored in the lower half of the *A*-register; the upper half of the *A*-register is all 0's.

k = 5: $(B)_j' \longrightarrow Y_L, Y_U$ is undisturbed. The complement of the contents of the *B*-register is stored in the lower half of the storage location at address $\bar{y}$; the upper half of the storage location is undisturbed.

k = 6: $(B)_j' \longrightarrow Y_U, Y_L$ is undisturbed. The complement of the contents of the *B*-register is transferred to the upper half of the storage location at address $\bar{y}$; the lower half of the storage location is undisturbed.

k = 7: $X\,(B)_j' \longrightarrow Y$. The complement of the contents of the *B*-register is transferred to the lower half of the storage location at address $\bar{y}$; the upper half of the storage location is an extension of the sign bit. The upper half of the storage location will be all 0's if bit 14 of the complement of the contents of the *B*-register is a 0 or it will be all 1's if bit 14 is a 1.

j DESIGNATORS: The selected *B*-register is specified as follows:

- j = 0: no operation is performed and the program advances to the next instruction.
- j = 1: *B*-register 1.
- j = 2: *B*-register 2.
- j = 3: *B*-register 3.
- j = 4: *B*-register 4.
- j = 5: *B*-register 5.
- j = 6: *B*-register 6.
- j = 7: *B*-register 7.

4. ARITHMETIC INSTRUCTIONS

Arithmetic instructions combine the contents of a storage location with the contents of a selected register or combine the contents of two registers forming a sum, difference, quotient, or product. The result that is formed is then stored in a storage location, retained in a register, or both.

SUBTRACTION

Subtraction in the UNIVAC 490 Real-Time Computer is binary subtraction with end-around borrow. An end-around borrow means that if a borrow is generated when the bit in position 29 of the subtrahend is subtracted from the bit in position 29 of the minuend, a binary 1 will be subtracted from the bit in position 0 of the difference. The following example illustrates this concept:

```
  00000000000000000000000000011   (a positive number)
- 11111111111111111111111111110   (a negative number)
  ----------------------------
  00000000000000000000000000101
                              1    (end around borrow)
  ----------------------------
  00000000000000000000000000100   (a positive number)
```

The following rules apply for subtraction:

1. If a negative number is subtracted from a positive number, the difference will be a positive number.
2. If a positive number is subtracted from a negative number, the difference will be a negative number.
3. If a positive number is subtracted from a positive number, the difference may be either a positive or a negative number.
4. If a negative number is subtracted from a negative number, the difference may be either a positive or a negative number.
5. If negative 0 is subtracted from positive 0, the difference will be positive 0.
6. If positive 0 is subtracted from negative 0, the difference will be negative 0.
7. If negative 0 is subtracted from negative 0, the difference will be positive 0.
8. If a number is subtracted from itself, the difference will be positive 0.

The above rules are followed except when the absolute value of the difference exceeds $2^{29}-1$. In this case, rule 1 or 2 is violated, as in the following examples:

```
  01001110001000111000011110 1100   (a positive number)
- 10001011000010110011101000 0000   (a negative number)
  ----------------------------
  11000011000110000100110110 1100
                              1    (end around borrow)
  ----------------------------
  11000011000110000100110110 1011   (a negative number)
```

The difference that is formed in this example is a negative number; consequently, it is an incorrect answer since it violates rule 1.

```
  10001101000100011010110001 1010   (a negative number)
- 01001110010111011100101001 1100   (a positive number)
  ----------------------------
  00111110101100101110000111 1110   (a positive number)
```

The difference that is formed in this example is a positive number; consequently, it is an incorrect answer since it violates rule 2. If there is a possibility that rule 1 or 2 for subtraction may be violated, the programmer is advised to make some provision in the program for handling these cases when they occur.

ADDITION

Addition in the UNIVAC 490 Real-Time Computer makes use of the subtraction process. Simply stated, addition is performed by complementing the addend and then subtracting it from the augend.

The following rules apply for addition:

1. If a negative number is added to a negative number, the sum will be a negative number.
2. If a positive number is added to a positive number, the sum will be a positive number.
3. If a negative number is added to a positive number, the sum may be either a positive or a negative number.
4. If a positive number is added to a negative number, the sum may be either a positive or a negative number.
5. If positive 0 is added to positive 0, the sum will be positive 0.
6. If negative 0 is added to negative 0, the sum will be negative 0.
7. If negative 0 is added to positive 0, the sum will be positive 0.
8. If a number is added to its complement, the sum will be positive 0.

The above rules are followed except when the absolute value of the sum exceeds $2^{29}_{10} - 1$. In these cases, rule 1 or 2 for addition is violated in the same manner that rule 1 or 2 for subtraction is violated.

If there is a possibility that rule 1 or 2 for addition may be violated, the programmer is advised to make some provision in the program for handling these cases when they occur.

ADD A

CLASS: Read

FUNCTION CODE: 20

MNEMONIC: ADD · *A*

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

DESCRIPTION: This instruction adds a 30-bit operand to the contents of the *A*-register, and retains the sum that is formed in the *A*-register.

k DESIGNATORS: The operand that is added to the contents of the *A*-register is derived as follows:

k = 0: $\overline{y} + (A) \longrightarrow A$. The lower half of the operand is $\overline{y}$, the low-order 15 bits contained in the instruction word after *B*-register modification; the upper half is all 0's.

k = 1: $Y_L + (A) \longrightarrow A$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's.

k = 2: $Y_U + (A) \longrightarrow A$. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's.

k = 3: $Y + (A) \longrightarrow A$. The operand is the 30-bits contained in the storage location at address $\overline{y}$.

k = 4: $X\overline{y} + (A) \longrightarrow A$. The lower half of the operand is $\overline{y}$—the low-order 15 bits contained in the instruction word after *B*-register modification; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 of $\overline{y}$ is a 0, or it will be all 1's if bit 14 is a 1.

k = 5: $XY_L + (A) \longrightarrow A$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half of the operand will be all 0's if bit 14 in storage location is a 0, or it will be all 1's if bit 14 is a 1.

k = 6: $XY_U + (A) \longrightarrow A$. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1.

k = 7: $(A) + (A) \longrightarrow A$. The operand is the 30-bits contained in the *A*-register

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if *(Q)* + or + 0.

j = 3: skip *NI* if *(Q)* is − or − 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip NI if $(A)_f$ is − or − 0.

SUBTRACT A

CLASS: Read

FUNCTION CODE: 21

MNEMONIC: SUB · *A*

OPERATION: $(A) - Y \longrightarrow A$

DESCRIPTION: This instruction subtracts a 30-bit operand from the contents of the *A*-register and retains the difference that is formed in the *A*-register.

k DESIGNATORS: The operand that is subtracted from the *A*-register is derived as follows:

k = 0: $(A) - \overline{y} \longrightarrow A$. The lower half of the operand is $\overline{y}$ − the low-order 15 bits contained in the instruction word after *B*-register modification; the upper half is all 0's.

k = 1: $(A) - Y_L \longrightarrow A$. The lower half of the operand is the low-order 15 bits contained in storage location at address $\overline{y}$; the upper half is all 0's.

k = 2: $(A) - Y_U \longrightarrow A$. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's.

k = 3: $A - Y \longrightarrow A$. The operand is the 30 bits contained in the storage location at address $\overline{y}$.

k = 4: $(A) - \overline{y} \longrightarrow A$. The lower half of the operand is $\overline{y}$ − the low-order 15 bits contained in the instruction word after *B*-register modification; the upper half is an extension of the sign bit. The upper half of the operand will be 0's if bit 14 of $\overline{y}$ is a 0, or it will be all 1's if bit 14 is a 1.

k = 5: $(A) - XY_L \longrightarrow A$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1.

k = 6: $(A) - XY_U \longrightarrow A$. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1.

k = 7: $(A) - (A) = +0 \longrightarrow A$. The operand is the 30 bits contained in the *A*-register. The operation that is performed is equivalent to subtracting a number from itself; consequently, the *A*-register will always contain + 0 when this value for *k* is used.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if *(Q)* is + or + 0.

j = 3: skip *NI* if *(Q)* is − or − 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is − or − 0.

ADD Q

CLASS: Read

FUNCTION CODE: 26

MNEMONIC: ADD · *Q*

OPERATION: $(Q) + Y \longrightarrow Q$

DESCRIPTION: This instruction adds a 30-bit operand to the contents of the *Q*-register and retains the sum that is formed in the *Q*-register.

k DESIGNATORS: The operand that is added to the contents of the Q-register is derived as follows:

k = 0: $\overline{y} + (Q) \longrightarrow Q$. The lower half of the operand is $\overline{y}$ – the low-order 15 bits are contained in the instruction word after B-register modification; the upper half is all 0's.

k = 1: $Y_L + (Q) \longrightarrow Q$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's.

k = 2: $Y_U + (Q) \longrightarrow Q$. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's.

k = 3: $Y + (Q) \longrightarrow Q$. The operand is the 30 bits contained in the storage location at address $\overline{y}$.

k = 4: $X\overline{y} + (Q) \longrightarrow Q$. The lower half of the operand is $\overline{y}$ – the low-order 15 bits contained in the instruction word after B-register modification; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 of $\overline{y}$ is a 0, or it will be all 1's if bit 14 is a 1.

k = 5: $XY_L + (Q) \longrightarrow Q$. The lower half of the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1.

k = 6: $XY_U + (Q) \longrightarrow Q$. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1.

k = 7: $(A) + (Q) \longrightarrow Q$. The operand is the 30 bits contained in the A-register.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip NI.

j = 2: skip NI if (A) is + or + 0.

j = 3: skip NI if (A) is – or – 0.

j = 4: skip NI if $(Q)_f$ is + 0.

j = 5: skip NI if $(Q)_f$ is not + 0.

j = 6: skip NI if $(Q)_f$ is + or + 0.

j = 7: skip NI if $(Q)_f$ is – or – 0.

SUBTRACT Q

CLASS: Read

FUNCTION CODE: 27

MNEMONIC: SUB · Q

OPERATION: $(Q) - Y \longrightarrow Q$

DESCRIPTION: This instruction subtracts a 30-bit operand from the contents of the Q-register and retains the difference that is formed in the Q-register.

k DESIGNATORS: The operand that is subtracted from the contents of the Q-register is derived as follows:

k = 0: $(Q) - \overline{y} \longrightarrow Q$. The lower half of the operand is $\overline{y}$ – the low-order 15 bits contained in the instruction word after B-register modification; the upper half is all 0's.

k = 1: $(Q) - Y_L \longrightarrow Q$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's.

k = 2: $(Q) - Y_U \longrightarrow Q$. The lower half of the operand is the high order bits contained in the storage location at address $\overline{y}$; the upper half is all 0's.

k = 3: $(Q) - Y \longrightarrow Q$. The operand is the 30 bits contained in the storage location at address $\overline{y}$.

k = 4: $(Q) - X\overline{y} \longrightarrow Q$. The lower half of the operand is $\overline{y}$ – the low-order 15 bits contained in the instruction word after B-register modification; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 of $\overline{y}$ is a 0, or it will be all 1's if bit 14 is a 1.

k = 5: $(Q) - XY_L \longrightarrow Q$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1.

k = 6: $(Q) - XY_U \longrightarrow Q$. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1.

k = 7: $(Q) - (A) \longrightarrow Q$. The operand is the 30 bits contained in the A-register.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if (A) is + or + 0.

j = 3: skip *NI* if (A) is − or − 0.

j = 4: skip *NI* if $(Q)_f$ is + 0.

j = 5: skip *NI* if $(Q)_f$ is not + 0.

j = 6: skip *NI* if $(Q)_f$ is + or + 0.

j = 7: skip *NI* if $(Q)_f$ is − or − 0.

ENTER Y + Q

CLASS: Read

FUNCTION CODE: 30

MNEMONIC: ENT · $Y + Q$

OPERATION: $Q + Y \longrightarrow A$

DESCRIPTION: This instruction adds a 30-bit operand to the contents of the Q-register and retains the sum that is formed in the A-register. The contents of the Q-register are undisturbed by this instruction.

k DESIGNATORS: The operand that is added to the contents of the Q-register is derived as follows:

k = 0: $(Q) + \overline{y} \longrightarrow A$. The lower half of the operand is $\overline{y}$ – the low-order 15 bits contained in the instruction word after B-register modification; the upper half is all 0's.

k = 1: $(Q) + Y_L \longrightarrow A$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's.

k = 2: $(Q) + Y_U \longrightarrow A$. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's.

k = 3: $(Q) + Y \longrightarrow A$. The operand is the 30 bits contained in the storage location at address $\overline{y}$.

k = 4: $(Q) + X\overline{y} \longrightarrow A$. The lower half of the operand is $\overline{y}$ – *the* low-order 15 bits contained in the instruction word after B-register modification; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 of $\overline{y}$ is a 0, or it will be all 1's if bit 14 is a 1.

k = 5: $(Q) + XY_L \longrightarrow A$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1.

k = 6: $(Q) + XY_U \longrightarrow A$. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all ones if bit 29 is a 1.

k = 7: $(Q) + (A) \longrightarrow A$. The operand is the 30 bits contained in the A-register.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if (Q) is + or + 0.

j = 3: skip *NI* if (Q) is − or − 0.

j = 4: skip *NI* if $(A)_f$ if + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is − or − 0.

ENTER Y – Q

CLASS: Read

FUNCTION CODE: 31

MNEMONIC: ENT · $Y - Q$

OPERATION: $Y - Q \longrightarrow A$.

DESCRIPTION: This instruction subtracts the contents of the Q-register from a 30-bit operand and retains the difference that is formed in the A-register. The contents of the Q-register are undisturbed by this instruction.

k DESIGNATORS: The operand that the contents of the register Q are subtracted from is derived as follows:

k = 0: $\bar{y} - (Q) \longrightarrow A$. The lower half of the operand is $\bar{y}$ – the low-order 15 bits contained in the instruction word after B-register modification; the upper half is all 0's.

k = 1: $Y_L - (Q) \longrightarrow A$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is all 0's.

k = 2: $Y_U - (Q) \longrightarrow A$. The lower half of the operand is the high-order 15 bits contained in storage location at address $\bar{y}$; the upper half is all 0's.

k = 3: $Y - (Q) \longrightarrow A$. The operand is the 30 bits contained in the storage location at address $\bar{y}$.

k = 4: $X\bar{y} - (Q) \longrightarrow A$. The lower half of the operand is $\bar{y}$ – the low order 15 bits contained in the instruction word after B-register modification; the upper half is an extension of the sign bit. The upper half of the operand will be 0's if bit 14 of $\bar{y}$ is a 0, or it will be all 1's if bit 14 is a 1.

k = 5: $XY_L - (Q) \longrightarrow A$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 in the storage location is a 0 or it will be all 1's if bit 14 is a 1.

k = 6: $XY_U - (Q) \longrightarrow A$. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 29 in the storage location is a 0 or it will be all 1's if bit 29 is a 1.

k = 7: $(A) - (Q) \longrightarrow A$. The operand is the 30 bits contained in the A-register.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if (Q) is + or + 0.

j = 3: skip *NI* if (Q) is – or – 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is – or – 0.

STORE A + Q

CLASS: Store

FUNCTION CODE: 32

MNEMONIC: STR · $A + Q$

OPERATION: $(A) + (Q) \longrightarrow Y$ and A

DESCRIPTION: The instruction adds the contents of the A- and Q-registers, retains the sum that is formed in the A-register, and stores this sum in a storage location.

k DESIGNATORS: The sum that is stored in the storage location is derived as follows:

k = 0: $(A) + (Q) \longrightarrow A, Q$. The sum is stored in the Q-register. Except when k is 0, the Q-register is undisturbed by this instruction.

k = 1: $(A) + (Q) \longrightarrow A$, $(A)_L \longrightarrow Y_L$, Y_U is undisturbed. The low-order 15 bits of the sum are stored in the lower half of the storage location at address $\bar{y}$; the upper half of the storage location is undisturbed.

k = 2: $(A) + (Q) \longrightarrow A$, $(A)_L \longrightarrow Y_U, Y_L$ is undisturbed. The low-order 15 bits of the sum are stored in the upper half of the storage location at address $\bar{y}$; the lower half of the storage location is undisturbed.

k = 3: $(A) + (Q) \rightarrow A, Y$. The sum is stored in the storage location at address $\bar{y}$.

k = 4: $(A) + (Q) \rightarrow A$. The sum is stored in the A-register.

k = 5: $(A) + (Q) \rightarrow A, (A)'_L \rightarrow Y_L, Y_U$ is undisturbed. The complement of the low-order 15 bits of the sum is stored in the lower half of the storage location at address $\bar{y}$; the upper half of the storage location is undisturbed.

k = 6: $(A) + (Q) \rightarrow A, (A)' \rightarrow Y_U, Y_L$ is undisturbed. The complement of the low-order 15 bits of the sum is stored in the upper half of the storage location at address $\bar{y}$; the lower half of the storage location is undisturbed.

k = 7: $(A) + (Q) \rightarrow A, (A)' \rightarrow Y$. The complement of the sum is stored in the storage location at address $\bar{y}$.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: *skip NI* if $(Q)_f$ is + or + 0.

j = 3: skip *NI* if $(Q)_f$ is − or − 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI if* $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is − or − 0.

STORE A − Q

CLASS: Store

FUNCTION CODE: 33

MNEMONIC: STR · $A - Q$

OPERATION: $(A) - (Q) \rightarrow Y$ and A

DESCRIPTION: This instruction subtracts the contents of the Q-register from the A-register, retains the difference that is formed in the A-register, and stores this difference in a storage location.

k DESIGNATORS: The difference that is stored in the storage location is derived as follows:

k = 0: $(A) - (Q) \rightarrow A, Q$. The difference is stored in the Q-register. With the exception of this value for k, the Q-register is undisturbed by this instruction.

k = 1: $(A) - (Q) \rightarrow A, (A)_L \rightarrow Y_L, Y_U$ is undisturbed. The low-order 15 bits of the difference are stored in the lower half of the storage location at address $\bar{y}$; the upper half of the storage location is undisturbed.

k = 2: $(A) - (Q) \rightarrow A, (A)_L \rightarrow Y_U, Y_L$ is undisturbed. The low-order 15 bits of the difference are stored in the upper half of the storage location at address $\bar{y}$; the lower half of the storage location is undisturbed.

k = 3: $(A) - (Q) \rightarrow A, Y$. The difference is stored in the storage location at address $\bar{y}$.

k = 4: $(A) - (Q) \rightarrow A$. The difference is stored in the A-register.

k = 5: $(A) - (Q) \rightarrow A, (A)'_L \rightarrow Y_L, Y_U$ is undisturbed. The complement of the low-order 15 bits of the difference is stored in the lower half of the storage location at address $\bar{y}$; the upper half of the storage location is undisturbed.

k = 6: $(A) - (Q) \rightarrow A, (A)'_L \; Y_U, Y_L$ is undisturbed. The complement of the low-order 15 bits of the difference is stored in the upper half of the storage location at address $\bar{y}$; the lower half of the storage location is undisturbed.

k = 7: $(A) - (Q) \rightarrow A, (A)' \rightarrow Y$. The complement of the difference is stored in the storage location at address $\bar{y}$.

j DESIGNATORS: The skip conditions are determined as follows:

j = 1: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if $(Q)_f$ is + or + 0.

j = 3: skip *NI* if $(Q)_f$ is − or − 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is − or − 0.

REPLACE A + Y

CLASS: Replace

FUNCTION CODE: 24

MNEMONIC: RPL · $A + Y$

OPERATION: $(A) + Y \longrightarrow Y$ and A

DESCRIPTION: This instruction adds a 30-bit operand to the contents of the A-register, retains the sum formed in the A-register, and stores this sum in the storage location from which the operand was obtained.

k DESIGNATORS: The operand that is added to the contents of the A-register and the sum stored in the storage location from which the operand was obtained are derived as follows:

k = 0, 4, or 7: not used.

k = 1: $Y_L + (A) \longrightarrow A$, $(A)_L \longrightarrow Y_L$; Y_U is undisturbed. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is all 0's. After the sum is formed in the A-register, the low-order 15 bits are stored in the lower half of the storage location from which the operand was obtained; the upper half of the storage location is undisturbed.

k = 2: $Y_U + (A) \longrightarrow A$, $(A)_L \longrightarrow Y_U$; Y_L is undisturbed. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is all 0's. After the sum is formed in the A-register, the low-order 15 bits are stored in the upper half of the storage location from which the operand was obtained; the lower half of the storage location is undisturbed.

k = 3: $Y + (A) \longrightarrow A$, Y. The operand is the 30 bits contained in the storage location at address $\bar{y}$. After the sum is formed in the A-register it is stored in the storage location from which the operand was obtained.

k = 5: $XY_L + (A) \longrightarrow A$, $(A)_L \longrightarrow Y_L$; Y_U is undisturbed. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1. After the sum is formed in the A-register, the low-order 15 bits of this sum are stored in the lower half of the storage location from which the operand was obtained; the upper half of the storage location is undisturbed.

k = 6: $XY_U + (A) \longrightarrow A$, $(A)_L \longrightarrow Y_U$; Y_L is undisturbed. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 29 in the storage location is 0, or it will be all 1's if bit 29 is a 1. After the sum is formed in the A-register, the low-order 15 bits of this sum are stored in the upper half of the storage location from which the operand was obtained; the lower half of the storage location is undisturbed.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip NI.

j = 2: skip NI if (Q) is + or + 0.

j = 3: skip NI if (Q) is − or − 0.

j = 4: skip NI if $(A)_f$ is + 0.

j = 5: skip NI if $(A)_f$ is not + 0.

j = 6: skip NI if $(A)_f$ is + or + 0.

j = 7: skip NI if $(A)_f$ is − or − 0.

REPLACE A − Y

CLASS: Replace

FUNCTION CODE: 25

MNEMONIC: RPL· $A - Y$

OPERATION: $(A) - Y \longrightarrow Y$ and A

DESCRIPTION: This instruction subtracts a 30-bit operand from the contents of the A-register, retains the difference that is formed in the A-register, and stores this sum in the storage location from which the operand was obtained.

k DESIGNATORS: The operand that is subtracted from the contents of the A-register and the difference that is stored in the storage location from which the operand was obtained are derived as follows:

k = 0, 4, or 7: not used.

k = 1: $(A) - Y_L \longrightarrow A$; $(A)_L \longrightarrow Y_L$; Y_U is undisturbed. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is all 0's. After the difference is formed in the A-register, the low-order 15 bits of this sum are stored in the lower half of the storage location from which the operand was obtained; the upper half of the storage location is undisturbed.

k = 2: $(A) - Y_U \longrightarrow A$, $(A)_L \longrightarrow Y_U$; Y_L is undisturbed. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is all 0's. After the difference is formed in the A-register, the low-order 15 bits of the difference are stored in the upper half of the storage location from which the operand was obtained.

k = 3: $(A) - Y \longrightarrow A, Y$. The operand is the 30 bits contained in the storage location at address $\bar{y}$. After the difference is formed in A-register, it is stored in the storage location from which the operand was obtained.

k = 5: $(A) - XY_L \longrightarrow A$, $(A)_L \longrightarrow Y_L$; Y_U undisturbed. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1. After the difference is formed in the A-register, the low-order 15 bits of this difference are stored in the lower half of the storage location from which the operand was obtained; the upper half of the storage location is undisturbed.

k = 6: $(A) - XY_U \longrightarrow A$, $(A)_L \longrightarrow Y_U$; Y_L is undisturbed. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1. After the difference is formed in the A-register, the low-order 15 bits of this difference are stored in the upper half of the storage location from which the operand was obtained; the lower half of the storage location is undisturbed.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if (Q) is + or + 0.

j = 3: skip *NI* if (Q) is − or − 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is − or − 0.

REPLACE Y + Q

CLASS: Replace

FUNCTION CODE: 34

MNEMONIC: RPL · $Y + Q$

OPERATION: $Y + (Q) \longrightarrow Y$ and A

DESCRIPTION: This instruction adds a 30-bit operand to the contents of the Q-register, retains the sum that is formed in the A-register, and stores this sum in the storage location from which the operand was obtained.

k DESIGNATORS: The operand that is added to the contents of the Q-register and the sum that is stored in the storage location that the operand was obtained from are derived as follows:

k = 0, 4, and 7: not used.

k = 1: $Y_L + (Q) \longrightarrow A$, $(A)_L \longrightarrow Y_L$; Y_U is undisturbed. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is all 0's. After the sum is formed in the A-register, the low-order 15 bits of this sum are stored in the lower half of the storage location from which the operand was obtained; the upper half of the storage location is undisturbed.

k = 2: $Y_U + (Q) \longrightarrow A$, $(A)_L \longrightarrow Y_U$; Y_L is undisturbed. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is all 0's. After the sum is formed in the A-register, the low-order 15 bits of this sum are stored in the upper half of the storage location from which the operand was obtained; the lower half of the storage location is undisturbed.

k = 3: $Y + (Q) \longrightarrow A, Y$. The operand is the 30-bits contained in the storage location at address $\overline{y}$. After the sum is formed in the *A*-register, it is stored in the storage location from which the operand was obtained.

k = 5: $XY_L + (Q) \longrightarrow A, (A)_L \longrightarrow Y_L ; Y_U$ is undisturbed. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1. After the sum is formed in the *A*-register, the low-order 15 bits of this sum are stored in the lower half of the storage location from which the operand was obtained; the upper half of the storage location is undisturbed.

k = 6: $XY_U + (Q) \longrightarrow A, (A)_L \longrightarrow Y_U ; Y_L$ is undisturbed. The lower half of the operand is the high order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 0's if bit 29 is a 0. After the sum is formed in the *A*-register, the low-order 15 bits of this sum are stored in the upper half of the storage location from which the operand was obtained; the lower half of the storage location is undisturbed.

j DESIGNATORS: The conditions for skip are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if (Q) is + or + 0.

j = 3: skip *NI* if (Q) is − or − 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is − or − 0.

REPLACE Y − Q

CLASS: Replace

FUNCTION CODE: 35

MNEMONIC: RPL · $Y - Q$

OPERATION. $Y - (Q) \longrightarrow Y$ and A

DESCRIPTION: This instruction subtracts the contents the *Q*-register from a 30-bit operand, retains the difference that is formed in the *A*-register, and stores this difference in the storage location from which the operand was obtained.

k DESIGNATORS: The operand that the contents of the *Q*-register are subtracted from and the difference that is stored in the storage location from which the operand was obtained are derived as follows:

k = 0, 4, and 7: not used.

k = 1: $Y_L - (Q) \longrightarrow A, (A)_L \longrightarrow Y_L ; Y_U$ is undisturbed. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's. After the difference is formed in the *A*-register the low-order 15 bits of this difference are stored in the lower half of the storage location from which the operand was obtained; the upper half of the storage location is undisturbed.

k = 2: $Y_U - (Q) \longrightarrow A, (A)_L \longrightarrow Y_U ; Y_L$ is undisturbed. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's. After the difference is formed in the *A*-register, the low-order 15 bits of this difference are stored in the upper half of the storage location from which the operand was obtained; the lower half of the storage location is undisturbed.

k = 3: $Y - (Q) \longrightarrow A, Y$. The operand is the 30 bits contained in the storage location at address $\overline{y}$: After the difference is formed in the *A*-register, it is stored in the storage location from which the operand was obtained from.

k = 5: $XY_L - (Q) \longrightarrow A, (A)_L \longrightarrow Y_L ; Y_U$ is undisturbed. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 of the storage location is a 0, or it will be all 1's if bit 14 is a 1. After the difference is formed in the *A*-register, the low-order 15 bits of this difference are stored in the lower half of the storage location from which the operand was obtained; the upper half of the storage location is undisturbed.

k = 6: XY_U − $(Q) \longrightarrow A$, $(A)_L \longrightarrow Y_U$; Y_L is undisturbed. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 29 of the storage location is a 0, or it will be all 1's if bit 29 is a 1. After the difference is formed in the *A*-register, the low-order 15 bits of this difference are stored in the upper half of the storage location from which the operand was obtained; the lower half of the storage location is undisturbed.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if (Q) is + or + 0.

j = 3: skip *NI* if (Q) is − or − 0.

j = 4: skip *NI* if (A_f) is + 0.

j = 5: skip *NI* if (A_f) is not + 0.

j = 6: skip *NI* if (A_f) is + or + 0.

j = 7: skip *NI* if (A_f) is − or − 0.

REPLACE Y + 1

CLASS: Replace

FUNCTION CODE: 36

MNEMONIC: RPL · *Y* + 1

OPERATION: $Y + 1 \longrightarrow Y$ and *A*

DESCRIPTION: This instruction adds a binary 1 to a 30-bit operand, retains the sum that is formed in the *A*-register, and stores this sum in the storage location from which the operand was obtained.

k DESIGNATORS: The operand that a binary 1 is added to and the sum that is stored in the storage location that the operand was obtained from are derived as follows:

k = 0, 4, and 7: not used.

k = 1: $Y_L + 1 \longrightarrow A$, $(A_L) \longrightarrow Y_L$; Y_U is undisturbed. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\overline{y}$, the upper half is all 0's. After the sum is formed in the *A*-register, the low-order 15 bits of this sum is stored in the lower half of the storage location from which the operand was obtained; the upper half of the storage location is undisturbed.

k = 2: $Y_U + 1 \longrightarrow A$, $(A)_L \longrightarrow Y_U$; Y_L is undisturbed. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's. After the sum is formed in the *A*-register, the low-order 15 bits of this sum is stored in the upper half of the storage location from which the operand was obtained; the lower half of the storage location is undisturbed.

k = 3: $Y + 1 \longrightarrow A$, Y. The operand is the 30-bits contained in the storage location at address $\overline{y}$. After the sum is formed in the *A*-register, it is stored in the storage location from which the operand was obtained.

k = 5: $XY_L + 1 \longrightarrow A$, $(A)_L \longrightarrow Y_L$; Y_U is undisturbed. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 of the storage location is a 0, or it will be all 1's if bit 14 is a 1. After the sum is formed in the *A*-register, the low-order 15 bits of this sum are stored in the lower half of the storage location from which the operand was obtained; the upper half of the storage location is undisturbed.

k = 6: $XY_U + 1 \longrightarrow A$, $(A)_L \longrightarrow Y_U$; Y_L is undisturbed. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 29 of the storage location is a 0, or it will be all 1's if bit 29 is a 1. After the sum is formed in the *A*-register, the low-order 15 bits of this sum are stored in the upper half of the storage location from which the operand was obtained; the lower half of the storage location is undisturbed.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if (Q) is + or + 0.

j = 3: skip *NI* if (Q) is − or − 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is − or − 0.

REPLACE Y − 1

CLASS: Replace

FUNCTION CODE: 37

MNEMONIC: RPL · *Y* − 1

OPERATION: $Y - 1 \longrightarrow Y$ and *A*

DESCRIPTION: This instruction subtracts a binary 1 from a 30-bit operand, retains the difference that is formed in the *A*-register, and stores this difference in the storage location from which the operand was obtained.

k DESIGNATORS: The operand that a binary 1 is subtracted from and the difference that is stored in the storage location from which the operand was obtained are derived as follows:

k = 0, 4, or 7: not used.

k = 0: $Y_L - 1 \longrightarrow A$, $(A)_L \longrightarrow Y_L$; Y_U is undisturbed. The lower half of the operand is the low order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's. After the difference is formed in the *A*-register, the low-order 15 bits of this difference are stored in the lower half of the storage location from which the operand was obtained; the upper half of the storage location is undisturbed.

k = 2: $Y_U - 1 \longrightarrow A$, $(A)_L \longrightarrow Y_U$; Y_L is undisturbed. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's. After the difference is formed in the *A*-register, the low-order 15 bits of this difference are stored in the upper half of the storage location from which the operand was obtained; the lower half of the storage location is undisturbed.

k = 3: $Y - 1 \longrightarrow A$, Y. The operand is the 30 bits contained in the storage location at address $\overline{y}$. After the difference is formed in the *A*-register, it is stored in the storage location from which the operand was obtained.

k = 5: $XY_L - 1 \longrightarrow A$, $(A)_L \longrightarrow Y_L$; Y_U is undisturbed. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 of the storage location is a 0, or it will be all 1's if bit 14 is a 1. After the difference is formed in the *A*-register, the low-order 15 bits of this difference are stored in the lower half of the storage location from which the operand was obtained; the upper half of the storage location is undisturbed.

k = 6: $XY_U - 1 \longrightarrow A$, $(A)_L \longrightarrow Y_U$; Y_L is undisturbed. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 29 of the storage location is a 0, or it will be all 1's if bit 29 is a 1. After the difference is formed in the *A*-register, the low-order 15 bits of this difference is stored in the upper half of the storage location from which the operand was obtained; the lower half of the storage location is undisturbed.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if (Q) is + or + 0.

j = 3: skip *NI* if (Q) is − or − 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is − or − 0.

MULTIPLICATION

Multiplication is performed with positive numbers, If a multiplication involves any negative numbers, they are made positive by complementing them prior to performing the multiplication. After the

positive product is formed in the AQ-register, the sign of this product is corrected by complementing the contents of the AQ-register, if one, but not both, of the original numbers was negative.

The following rules apply for multiplication:

- If a positive number is multiplied by a positive number or a negative number by a negative number, the product will be positive.
- If a positive number is multiplied by a negative number or a negative number by a positive number, the product will be negative.
- If positive 0 is multiplied by positive 0 or negative 0 by negative 0, the product will be positive 0.
- If positive 0 is multiplied by negative 0 or negative 0 by positive 0, the product will be negative 0.

If the range of the multiplicands and multipliers is known, it is possible to determine if the entire product will be in the Q-register or it part of it will be in the A-register by applying the following rules:

- If the most significant bit (the most significant 1 in a positive number or the most significant 0 in a negative number) of the multiplicand is in bit position n, the entire product will be in the Q-register if bit position 28-n of the multiplier contains a sign bit.
- If the most significant bit of the multiplicand is in bit position n, the entire product may be in the Q-register if the most significant bit of the multiplier is in bit position 28-n.
- If the most significant bit of the multiplicand is in bit position n, the entire product will not be in the Q-register if bit position 29-n of the multiplier does not contain a sign bit.

The following examples illustrate these rules:

- The entire product will be contained in the Q-register.

```
                                  0000000000000000000000001010  (multiplicand)
                                  0000000000000000000000001000  (multiplier)
00000000000000000000000000000     0000000000000000000000001010000  (product)
      in the A-register                    in the Q-register
```

- The entire product may be contained in the Q-register.

```
                                  0000000000000000000000000010  (multiplicand)
                                  0010000000000000000000000000  (multiplier)
00000000000000000000000000000     0100000000000000000000000000  (product)
      in the A-register                    in the Q-register

                                  0011111111111111111111111111  (multiplicand)
                                  0000000000000000000000000011  (multiplier)
00000000000000000000000000000     1011111111111111111111111101  (product)
      in the A-register                    in the Q-register
```

- The entire product will not be contained in the Q-register.

```
                                  0111111111111111111111111111  (multiplicand)
                                  1000000000000000000000000000  (multiplier)
11000000000000000000000000000     1111111111111111111111111110  (product)
      in the A-register                    in the Q-register
```

MULTIPLY

CLASS: Read

FUNCTION CODE: 22

MNEMONIC: MUL

OPERATION: $(Q) \times Y \longrightarrow AQ$

DESCRIPTION: This instruction multiplies the contents of the Q-register by a 30-bit multiplier and retains the 60-bit product that is formed in the AQ-register. The interpretation of the j designator is made prior to final sign correction.

k DESIGNATORS: The multiplier that the contents of the Q-register are multiplied by is derived as follows:

k = 0: $y \times (Q) \longrightarrow AQ$. The lower half of the multiplier is $\overline{y}$ – the low-order 15 bits contained in the instruction word after B-register modification; the upper half is all 0's.

k = 1: $Y_L \times (Q) \longrightarrow AQ$. The lower half of the multiplier is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's.

k = 2: $Y_U \times (Q) \longrightarrow AQ$. The lower half of the multiplier is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's.

k = 3: $Y \times (Q) \longrightarrow AQ$. The multiplier is the 30-bits contained in the storage location at address $\overline{y}$.

k = 4: $X\bar{y}$ x $(Q) \rightarrow AQ$. The lower half of the multiplier is $\bar{y}$ – the low-order 15 bits contained in the instruction word after *B*-register modification; the upper half is an extension of the sign bit. The upper half of the multiplier will be all 0's if bit 14 of $\bar{y}$ is a 0, or it will be all 1's if bit 14 is a 1.

k = 5: XY_L x $(Q) \rightarrow AQ$. The lower half of the multiplier is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is an extension of the sign bit. The upper half of the multiplier will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1.

k = 6: XY_U x $(Q) \rightarrow AQ$. The lower half of the multiplier is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is an extension of the sign bit. The upper half of the multiplier will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1.

k = 7: (A) x $(Q) \rightarrow AQ$. The multiplier is the 30 bits contained in the *A*-register. This value for *k* may be used if $(Q)_i$ is a positive number. If $(Q)_i$ is a negative number, a complemented product will result.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if (Q) is + or + 0 prior to final sign correction, If a skip does not occur, a double length product is indicated since there is a significant bit in bit position 29 of the *Q*-register.

j = 3: skip *NI* if (Q) is – or – 0 prior to final sign correction. If a skip occurs, a double length product is indicated since there is a significant bit in bit position 29 of the *Q*-register.

j = 4: skip *NI* if (A) is + 0 prior to final sign correction. If a skip occurs, it indicates that the product has 30 or less significant bits, and that the *A*-register contains only sign bits. This does not mean that the *Q*-register contains the correct product, since bit position 29 of the *Q*-register may contain a significant bit of the product, thus making bit position 0 of the *A*-register the first sign bit. If a skip does not occur, it indicates that significant bits of the product are in the *A*-register.

j = 5: skip *NI* if (A) is not + 0 prior to final sign correction. If a skip occurs, it indicates that significant bits of the product are in the *A*-register. If a skip does not occur, it indicates the same situation that is present when a skip occurs with *j* = 4.

j = 6: skip *NI* if (A) is + or + 0 prior to final sign correction. A Skip should always occur since (A) should always be a positive number prior to final sign correction.

j = 7: skip *NI* is (A) is – or – 0 prior to final sign correction. A skip should never occur since (A) should never be a negative number prior to final sign correction.

DIVISION

Division is also performed with positive numbers. If a division involves any negative numbers, they are made positive by complementing them prior to performing the division. After the positive quotient and remainder are formed in the *Q*- and *A*-register respectively, the signs of the quotient and remainder are corrected by complementing the contents of the *Q*- and *A*-registers if one, but not both, of the original numbers was negative.

The following rules apply for division:

- If a positive number is divided by a positive number or a negative number by a negative number, the quotient and remainder will be positive numbers.
- If a positive number is divided by a negative number or a negative number by a positive number, the quotient and remainder will be negative numbers.

Negative Zero Quotients and Remainders

Division, if handled improperly, may generate a negative 0 quotient or remainder that can have an adverse affect on further calculations. This situation can occur in the following cases:

Even Division Where The Dividend and Divisor Have Unlike Signs and Are Non-Zero

The result of such a division is that the correct quotient will be in the *Q*-register and the remainder in the *A*-register will be a negative 0.

When the interpretation of the *j*-designator is made, the *Q*-register will contain the absolute value of the quotient and the *A*-register will contain a positive 0. For example:

```
00000000000000000000000000000000  00000000000000000010100111001101 (dividend)
                                  11111111111111111110101100011010 (divisor)
                                  ------------------------------------
                                  00000000000000000000000000000010 (quotient in the Q-register)
At j interpretation               00000000000000000000000000000000 (remainder in the A-register)

                                  11111111111111111111111111111101 (quotient in the Q-register)
Final Result                      11111111111111111111111111111111 (remainder in the A-register)
```

Division Where The Absolute Value Of The Divisor Is Greater Than The Absolute Value Of Dividend, The Signs Are Unlike, And Both Are Non-Zero

When division is performed in this case, the quotient in the *Q*-register will be a negative 0 and the remainder in the *A*-register will be the complement of the absolute value of the dividend. At the time the interpretation of the *j*-designator is made, the *Q*-register will contain a positive 0 and the *A*-register will contain the absolute value of the dividend. For example:

```
00000000000000000000000000000000  00000000000000000000000000000011 (dividend)
                                  11111111111111111111111111111010 (divisor)
                                  ------------------------------------
                                  00000000000000000000000000000000 (quotient in the Q-register)
At j interpretation               00000000000000000000000000000011 (remainder in the A-register)

                                  11111111111111111111111111111111 (quotient in the Q-register)
Final Result                      11111111111111111111111111111100 (remainder in the A-register)
```

Division By Positive Or Negative Zero

The following rules apply in these cases:

- If a positive number is divided by positive 0, the quotient in the *Q*-register will be a negative 0 and the remainder in the *A*-register will be the initial contents of the *Q*-register. At the time interpretation of the *j*-designator is made, the contents of the *Q*- and *A*-registers will be the same as the final contents.

- If a negative number is divided by positive 0, the quotient in the *Q*-register will be a positive 0 and the remainder in the *A*-register will be the initial contents of the *Q*-register. At the time interpretation of the *j*-designator is made, the *Q*-register will contain a negative 0 and the *A*-register will contain the complement of the initial contents of the *Q*-register.

- If a positive number is divided by negative 0, the quotient in the *Q*-register will be a positive 0 and the remainder in the *A*-register will be the complement of the initial contents of the *Q*-register. At the time the *j*-designator is interpreted, the *Q*-register will contain a negative 0 and the *A*-register will contain the initial contents of the *Q*-register.

- If a negative number is divided by negative 0, the quotient in the *Q*-register will be a negative 0 and the remainder in the *A*-register will be the complement of the initial contents of the *Q*-register. At the time the *j*-designator is interpreted, the contents of the *Q*- and *A*-registers will be the same as the final contents

The following examples illustrate these rules:

- A positive number divided by positive 0.

```
00000000000000000000000000000001  00000000000000000000000000000001 (dividend)
                                  00000000000000000000000000000000 (divisor)
                                  ------------------------------------
                                  11111111111111111111111111111111 (quotient in the Q-register)
At j interpretation               00000000000000000000000000000001 (remainder in the A-register)

                                  11111111111111111111111111111111 (quotient in the Q-register)
Final Result                      00000000000000000000000000000001 (remainder in the A-register)
```

- A negative number divided by positive 0.

```
11111111111111111111111111111111  11111111111111111111111111111110 (dividend)
                                  00000000000000000000000000000000 (divisor)
                                  ------------------------------------
                                  11111111111111111111111111111111 (quotient in the Q-register)
At j interpretation               00000000000000000000000000000001 (remainder in the A-register)

                                  00000000000000000000000000000000 (quotient in the Q-register)
Final Result                      11111111111111111111111111111110 (remainder in the A-register)
```

- A positive number divided by negative 0.

```
01011111111111111111111111111111  11011111111111111111111111111111 (dividend)
                                  11111111111111111111111111111111 (divisor)
                                  ------------------------------------
                                  11111111111111111111111111111111 (quotient in the Q-register)
At j interpretation               11011111111111111111111111111111 (remainder in the A-register)

                                  00000000000000000000000000000000 (quotient in the Q-register)
Final Result                      00100000000000000000000000000000 (remainder in the A-register)
```

- A negative number divided by negative 0.

```
10011111111111111111111111111111  11011111111111111111111111111111 (dividend)
                                  11111111111111111111111111111111 (divisor)
                                  ------------------------------------
                                  11111111111111111111111111111111 (quotient in the Q-register)
At j interpretation               00100000000000000000000000000000 (remainder in the Q-register)

                                  11111111111111111111111111111111 (quotient in the Q-register)
Final Result                      00100000000000000000000000000000 (remainder in the A-register)
```

Division Of Positive Or Negative Zero By a Non-Zero Divisor With An Unlike Sign

When division is performed in this case, the quotient in the Q-register and the remainder in the A-register will be a negative 0. At the time interpretation of the j-designator is made both the Q- and A-register will contain a positive 0. The following examples will illustrate this:

```
00000000000000000000000000000000  00000000000000000000000000000000 (dividend)
                                  11111111111111111111111111111110 (divisor)
                                  ---------------------------------
At j interpretation               { 00000000000000000000000000000000 (quotient in the Q-register)
                                  { 00000000000000000000000000000000 (remainder in the A-register)
Final Result                      { 11111111111111111111111111111111 (quotient in the Q-register)
                                  { 11111111111111111111111111111111 (remainder in the A-register)

11111111111111111111111111111111  11111111111111111111111111111111 (dividend)
                                  00000000000000000000000000000001 (divisor)
                                  ---------------------------------
At j interpretation               { 00000000000000000000000000000000 (quotient in the Q-register)
                                  { 00000000000000000000000000000000 (remainder in the A-register)
Final Result                      { 11111111111111111111111111111111 (quotient in the Q-register)
                                  { 11111111111111111111111111111111 (remainder in the A-register)
```

Divide Overflow With Non-Zero Divisor and Dividend

In divison, the dividend in the AQ-register may have up to 59 significant bits while the divisor may have as few as 1. In these cases, a quotient may be generated that has as many as 59 significant bits. Since the Q-register has a 30-bit capacity, an overflow situation will result when a quotient is generated that has more than 29 significant bits. If overflow does occur, the quotient in the Q-register will be a positive 0 if the divisor and dividend have unlike signs, or it will be a negative 0 if the signs were the same. At the time the j-designator is interpreted the Q-register will always contain a negative 0.

The following rules govern the occurrence of a divide overflow:

- If the most significant bit of the divisor is in bit position n, a divide overflow will not occur if the dividend has no significant bits beyond bit position n + 28.
- If the most significant bit of the divisor is in bit position n, a divide overflow will occur if the dividend has a significant bit in bit position n + 30 or beyond.
- If the most significant bit of the divisor is in bit position n, a divide overflow may occur if the most significant bit of the dividend is in bit position n + 29.

The following examples illustrate these rules:

- No overflow.

```
00000000000000000000000000000000  00000000000000001011000011100 (dividend)
                                  00000000000000000001010011100 (divisor)
                                  ---------------------------------
At j interpretation               { 00000000000000000000000000010001 (quotient in the Q-register)
                                  { 00000000000000000000000000000000 (remainder in the A-register)
Final Result                      { 00000000000000000000000000010001 (quotient in the Q-register)
                                  { 00000000000000000000000000000000 (remainder in the A-register)
```

- Overflow occurs.

```
00000000000000010001001100011  01001111011110011110011010111 (dividend)
                               00000000000000010000001000000 (divisor)
                               ---------------------------------
At j interpretation            { 11111111111111111111111111111111 (quotient in the Q-register)
                               { 01001111011110001100111101011 (remainder in the A-register)
Final Result                   { 11111111111111111111111111111111 (quotient in the Q-register)
                               { 01001111011110001100111101011 (remainder in the A-register)
```

- Overflow may occur.

```
11111111111111110111001010111100  10110000100011010001100111111 (dividend)
                                  00010000000000010000001000000 (divisor)
                                  ---------------------------------
At j interpretation               { 11111111111111111111111111111111 (quotient in the Q-register)
                                  { 01001111011110011110011000000 (remainder in the Q-register)
Final Result                      { 00000000000000000000000000000000 (quotient in the Q-register)
                                  { 10110000100011010001100111111 (remainder in the A-register)
```

In this example, overflow occurs.

```
00000000000000000000000000000001  00000000000000000000000000000000 (dividend)
                                  00000000000000000000000000000011 (divisor)
                                  ---------------------------------
At j interpretation               { 00001000100010001000011110110 (quotient in the Q-register)
                                  { 00000000000000000000000000000001 (remainder in the A-register)
Final Result                      { 00001000100010001000011110110 (quotient in the Q-register)
                                  { 00000000000000000000000000000001 (remainder in the A-register)
```

In this example, overflow does not occur.

The remainder in overflow division is difficult to determine and the value of such information, when obtained, is questionable. The rules that are stated below are valid at least in the above examples. They should not, however, be considered universal rules.

- If the dividend and divisor are positive numbers, add the dividend and divisor. The remainder in the A-register will be the low-order 30 bits of the sum that is formed. At the time the j-designator is interpreted, the contents of the A-register will be the same as the final contents.

- If the dividend and divisor are negative numbers, complement the dividend and divisor, and then add them. The remainder in the *A*-register will be the low-order 30 bits of the sum that is formed. At the time interpretation of the *j*-designator is made, the contents of the *A*-register will be the same as the final contents.
- If the dividend is a positive number and the divisor is a negative number, the divisor should be complemented and then added to the dividend. The final remainder in the A-register will be the complement of the low-order 30 bits of the sum that is formed. At the time interpretation of the *j*-designator is made, the contents of the *A*-register will be the low-order 30 bits of the sum that is formed.
- If the dividend is a negative number and the divisor is a positive number, complement the dividend should be complemented and then added it to the divisor. The final remainder in the *A*-register will be the complement of the low-order 30 bits of the sum that is formed. At the time the *j*-designator is interpreted, the contents of the *A*-register will be the low-order 30 bits of the sum that is formed.

DIVIDE

CLASS: Read

FUNCTION CODE: 23

MNEMONIC: DIV.

OPERATION: $(AQ) \div Y \rightarrow Q$, Remainder$\rightarrow A$

DESCRIPTION: This instruction divides the contents of the *AQ*-register by a 30–bit divisor and retains the quotient and remainder that are formed in the *Q*- and *A*-registers respectively. The interpretation of the *j*-designator is made prior to final sign correction.

k DESIGNATORS: The divisor that the contents of the *AQ*-register are divided by is derived as follows:

k = 0: $(AQ) \div \overline{y} \rightarrow Q$, $R \rightarrow A$. The lower half of the divisor is $\overline{y}$ – the low-order 15 bits contained in the instruction word after *B*-register modification; the upper half is all 0's.

k = 1: $(AQ) \div Y_L \rightarrow Q$, $R \rightarrow A$. The lower half of the divisor is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's.

k = 2: $(AQ) \div Y_U \rightarrow Q$, $R \rightarrow A$. The lower half of the divisor is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's.

k = 3: $(AQ) \div Y \rightarrow Q$, $R \rightarrow A$. The divisor is the 30 bits contained in the storage location at address $\overline{y}$.

k = 4: $(AQ) \div X\overline{y} \rightarrow Q$, $R \rightarrow A$. The lower half of the divisor is $\overline{y}$ – the low-order 15 bits contained in the instruction word after *B*-register modification; the upper half is an extension of the sign bit. The upper half of the divisor will be all 0's if bit 14 of $\overline{y}$ is a 0, or it will be all 1's if bit 14 is a 1.

k = 5: $(AQ) \div XY_L \rightarrow Q$, $R \rightarrow A$. The lower half of the divisor is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the divisor will be all 0's if bit 14 of the storage location is a 0, or it will be all 1's if bit 14 is a 1.

k = 6: $(AQ) \div XY_U \rightarrow Q$, $R \rightarrow A$. The lower half of the divisor is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the divisor will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1.

k = 7: not used.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if *(Q)* is + or + 0 prior to final sign correction. If a skip does not occur, a faulty divide is indicated.

j = 3: skip *NI* if *(Q)* is – or – 0 prior to final sign correction. If a skip occurs, a faulty divide is indicated.

j = 4: skip *NI* if *(A)* is + 0 prior to final sign correction. If a skip occurs, no remainder is indicated.

j = 5: skip *NI* if *(A)* is not + 0 prior to final sign correction. If a skip occurs, a remainder is indicated.

j = 6: skip *NI* if *(A)* is + or + 0 prior to final sign correction. If a skip occurs, a legitimate divide with or without remainder is indicated.

j = 7: skip *NI* if *(A)* is – or – 0 prior to final sign correction. If a skip occurs, a faulty divide is indicated.

COMPARE

CLASS: Read

FUNCTION CODE: 04

MNEMONIC: COM . *A* . *Q*, . *AQ*

OPERATION: Compare *(A)* and/or *(Q)* with *Y* to determine skip.

DESCRIPTION: This instruction compares the signed value of a 30–bit operand with the signed value of the contents of the *A*-register and/or the contents of the *Q*-register.

k DESIGNATORS: The operand that the contents of the *A*-register and/or the *Q*-register are compared with is derived as follows:

k = 0: $Y = \overline{y}$. The lower half of the operand is $\overline{y}$ – the low-order 15 bits contained in the instruction word after *B*-register modification; the upper half is all 0's.

k = 1: $Y = Y_L$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's.

k = 2: $Y = Y_U$. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's.

k = 3: $Y = Y$. The operand is the 30 bits contained in the storage location at address $\overline{y}$.

k = 4: $Y = X\overline{y}$. The lower half of the operand is $\overline{y}$ – the low-order 15 bits contained in the instruction word after *B*-register modification; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 of $\overline{y}$ is a 0, or it will be all 1's if bit 14 is a 1.

k = 5: $Y = XY_L$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 of the storage location is a 0, or it will be all 1's if bit 14 is a 1.

k = 6: $Y = XY_U$. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 29 of the storage location is a 0, or it will be all 1's if bit 29 is a 1.

k = 7: $Y = (A)$. The operand is the 30 bits contained in the *A*-register.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if $Y \leq (Q)$.

j = 3: skip *NI* if $Y > (Q)$.

j = 4: skip *NI* if $(A) < Y \leq (Q)$.

j = 5: skip *NI* if $Y > (Q)$ or if $Y \leq (A)$.

j = 6: skip *NI* if $Y \leq (A)$.

j = 7: skip *NI* if $Y > (A)$.

Negative Zero in Compare Operations

Negative 0 has the following properties in compare operations;

1. $-0 = -0$.
2. $-0 \neq +0$.
3. $-0 < +0$.
4. $-Y < -0$ if $Y \neq 0$.

To summarize, if $Y \neq 0$, $-Y < -0 < +0 < +Y$.

5. SHIFT INSTRUCTIONS

Shift instructions shift the contents of a selected register, Y bit positions right or left. The shift count, Y, is always treated as a 6–bit positive number regardless of the configuration of the other 24 bits.

RIGHT SHIFT Q

CLASS: Read

FUNCTION CODE: 01

MNEMONIC: RSH . Q

OPERATION: Shift (Q) right Y bit positions.

DESCRIPTION: This instruction shifts the contents of register Q, Y bit positions to the right. The bits that are shifted off the right end are lost and sign bits fill in on the left end in those bit positions that were vacated. If $Y > 59_{10}$, an incorrect shift will result. If $29 \leqslant Y \leqslant 59_{10}$, all bit positions of the Q-register will contain the original sign bit.

k DESIGNATORS: The shift count, Y, is determined as follows:

k = 0 or 4: $Y = \overline{y}$. The shift count is the low-order 6 bits of $\overline{y}$; that is, the low-order 6 bits contained in the instruction word after B-register modification.

k = 1, 3, or 5: $Y = Y_L$. The shift count is the low-order 6 bits contained in the lower half of the storage location at address $\overline{y}$.

k = 2 or 6: $Y = Y_U$. The shift count is the low-order 6 bits contained in the upper half of the storage location at address $\overline{y}$.

k = 7: $Y = A_L$. The shift count is the low-order 6 bits contained in the A-register.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if $(Q)_f$ is + or + 0.

j = 3: skip *NI* if $(Q)_f$ is – or – 0.

j = 4: skip *NI* if (A) if + 0.

j = 5: skip *NI* if (A) is not + 0.

j = 6: skip *NI* if (A) is + or + 0.

j = 7: skip *NI* if (A) is – or – 0.

Examples:

1. $(Q)_i$ = 100100101111111011100001110101
   $Y = 8_{10}$
   $(Q)_f$ = 111111111001001011111110111000

2. $(Q)_i$ = 001111111101111011110001110111
   $Y = 29_{10}$
   $(Q)_f$ = 000000000000000000000000000000

RIGHT SHIFT A

CLASS: Read

FUNCTION CODE: 02

MNEMONIC: RSH . *A*

OPERATION: Shift *(A)* right *Y* bit positions,

DESCRIPTION: This instruction shifts the contents of the *A*-register *Y* bit positions to the right. The bits that are shifted off the right end are lost and sign bits fill in on the left end in those bit positions that were vacated. If $Y > 59_{10}$, an incorrect shift will result. If $29 \leqslant Y \leqslant 59_{10}$, all bit positions of the *A*-register will contain the original sign bit.

k DESIGNATORS: The shift count, *Y*, is derived as follows:

k = 0 or 4: $Y = \bar{y}$. The shift count is the low-order 6 bits of $\bar{y}$; that is, the low-order 6 bits contained in the instruction word after *B*-register modification.

k = 1, 3, or 5: $Y = Y_L$. The shift count is the low-order 6–bits contained in the lower half of the storage location at address $\bar{y}$.

k = 2 or 6: $Y = Y_U$. The shift count is the low-order 6 bits contained in the upper half of the storage location at address $\bar{y}$.

k = 7: $Y = A_L$. The shift count is the low-order 6 bits contained in the *A*-register.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if *(Q)* is + or + 0.

j = 3: skip *NI* if *(Q)* is – or – 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is – or – 0.

RIGHT SHIFT AQ

CLASS: Read

FUNCTION CODE: 03

MNEMONIC: RSH . *AQ*

OPERATION: Shift *(AQ)* right *Y* bit positions.

DESCRIPTION: This instruction shifts the contents of the *AQ*-register, *Y* bit positions to the right. The bits that are shifted off the right end of the *Q*-register are lost, the bits that are shifted off the right end of the *A*-register fill in on the left end of the *Q*-register in those positions that were vacated, and the left end of the *A*-register is filled in with sign bits. (The sign bit in this case is the bit in bit position 29 of the *A*-register.) If $Y > 59_{10}$, an incorrect shift will result. If $Y \geq 29_{10}$, all bit positions of the *A*-register will contain a sign bit. If $Y = 59_{10}$, all bit positions of both the *A*- and *Q*-registers will contain a sign bit.

k DESIGNATORS: The shift count, *Y*, is derived as follows:

k = 0 or 4: $Y = \bar{y}$. The shift count is the low-order 6 bits of $\bar{y}$; that is, the low-order 6 bits contained in the instruction word after *B*-register modification.

k = 1, 3, or 5: $Y = Y_L$. The shift count is the low-order 6 bits contained in the lower half of the storage location at address $\bar{y}$.

k = 2 or 6: $Y = Y_U$. The shift count is the low-order 6 bits contained in the upper half of the storage location at address $\bar{y}$.

k = 7: $Y = A_L$. The shift count is the low-order 6 bits contained in the *A*-register.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if $(Q)_f$ is + or + 0.

j = 3: skip *NI* if $(Q)_f$ is – or – 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is – or – 0.

Examples:

	(A)	(Q)
1. $(AQ)_i$ =	11101001111111111111011010001100	00111010110101001010001100 1001
$Y = 12_{10}$	(A)	(Q)
$(AQ)_f$ =	11111111111111110100111111111111	01101000110000111010110101001 0
	(A)	(Q)
2. $(AQ)_i$ =	10111100001110101000000000000001	11111111110000010101010101010 1
$Y = 59_{10}$	(A)	(Q)
$(AQ)_f$ =	11111111111111111111111111111111	11111111111111111111111111111111

LEFT SHIFT Q

CLASS: Read

FUNCTION CODE: 05

MNEMONIC: LSH . *Q*

OPERATION: Shift (Q) left *Y* bit positions, end around.

DESCRIPTION: This instruction shifts the contents of the *Q*-register, *Y* bit positions to the left. The bits that are shifted off the left end fill in on the right end in the bit positions that were vacated. If $Y > 59_{10}$, an incorrect shift will result. If $Y = 30_{10}$, the *Q*-register will be restored to its initial condition.

k DESIGNATORS: The shift count, *Y*, is derived as follows:

k = 0 or 4: $Y = \bar{y}$. The shift count is the low-order 6 bits of $\bar{y}$; that is, the low-order 6 bits contained in the instruction word after *B*-register modification.

k = 1, 3, or 5: $Y = Y_L$. The shift count is the low-order 6 bits contained in the lower half of the storage location at address $\bar{y}$.

k = 2 or 6: $Y = Y_U$. The shift count is the low-order 6 bits contained in the upper half of the storage location at address $\bar{y}$.

k = 7: $Y = A_L$. The shift count is the low-order 6 bits contained in the *A*-register.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip

j = 1: skip *NI*.

j = 2: skip *NI* if $(Q)_f$ is + or + 0.

j = 3: skip *NI* if $(Q)_f$ is − or − 0.

j = 4: skip *NI* if *(A)* is + 0.

j = 5: skip *NI* if *(A)* is not + 0.

j = 6: skip *NI* if *(A)* is + or + 0.

j = 7: skip *NI* if *(A)* is − or − 0.

Examples:

1. $(Q)_i$ = 00111010110101001010001100 1001

   $Y = 15_{10}$

   $(Q)_f$ = 01010001100100100111010110 1010

2. $(Q)_i$ = 11101001111111111111011010001100

   $Y = 30_{10}$

   $(Q)_f$ = 11101001111111111111011010001100

LEFT SHIFT A

CLASS: Read

FUNCTION CODE: 06

MNEMONIC: LSH . *A*

OPERATION: Shift *(A)* left *Y* bit positions, end around.

DESCRIPTION: This instruction shifts the contents of the *A*-register *Y* bit positions to the left. The bits that are shifted off the left end fill in on the right end in the bit positions that were vacated. If $Y > 59_{10}$, an incorrect shift will result. If $Y = 30_{10}$, the *A*-register will be restored to its initial condition.

k DESIGNATORS: The shift count, *Y*, is derived as follows:

k = 0 or 4: $Y = \bar{y}$. The shift count is the low-order 6 bits of $\bar{y}$; that is, the low-order 6 bits contained in the instruction word after *B*-register modification.

k = 1, 3, or 5: $Y = Y_L$. The shift count is the low-order 6 bits contained in the lower half of the storage location at address $\bar{y}$.

k = 2 or 6: $Y = Y_U$. The shift count is the low-order 6 bits contained in the upper half of the storage location at address $\bar{y}$.

k = 7: $Y = A_L$. The shift count is the low-order 6 bits contained in the *A*-register.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if *(Q)* is + or + 0.

j = 3: skip *NI* if *(Q)* is − or − 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is − or − 0.

LEFT SHIFT AQ

CLASS: Read

FUNCTION CODE: 07

MNEMONIC: *LSH . AQ*

OPERATION: Shift *(AQ)* left *Y* bit positions, end around.

DESCRIPTION: This instruction shifts the contents of the *AQ*-register *Y* bit positions to the left. The bits that are shifted off the left end of the *A*-register fill in on the right end of the *Q*-register in the bit positions that were vacated, and the bits that are shifted off the left end of register *Q* fill in similarly on the right end of the *A*-register. If $Y > 59_{10}$, an incorrect shift will result. If $Y = 30_{10}$, the contents of the *A*-register and the *Q*-register will be interchanged.

k DESIGNATORS: The shift count, *Y*, is derived as follows:

k = 0 or 4: $Y = \bar{y}$. The low-order 6 bits of $\bar{y}$; that is, the low-order 6 bits contained in the instruction word after *B*-register modification.

k = 1, 3, or 5: $Y = Y_L$. The low-order 6 bits contained in the lower half of the storage location at address $\bar{y}$.

k = 2 or 6: $Y = Y_U$. The low-order 6 bits contained in the upper half of the storage location at address $\bar{y}$.

k = 7: $Y = A_L$. The low-order 6 bits contained in the *A*-register.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if $(Q)_f$ is + or + 0.

j = 3: skip *NI* if $(Q)_f$ is − or − 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is − or − 0.

Examples:

	(A)	(Q)
1. $(AQ)_i$ =	111010011111111111011010001100	001110101101010010100011001001
$Y = 6_{10}$	(A)	(Q)
$(AQ)_f$ =	011111111111101101000110000111 0	101101010010100011001001111010
	(A)	(Q)
2. $(AQ)_i$ =	101111101010001100010011101101	000000011111111101010100011001
$Y = 30_{10}$	(A)	(Q)
$(AQ)_f$ =	000000011111111101010100011001	101111101010001100010011101101

6. LOGICAL INSTRUCTIONS

Logical instructions are based upon five operations: logical product, selective set, selective clear, selective complement, and selective substitute. They operate on the individual bit positions of the operands; that is, the condition of a specific bit position in the result is determined by the condition of the corresponding bit positions in the operands.

LOGICAL PRODUCT, L [Y(Q)]

Logical product operations are generally used for masking; that is, lifting out a selected portion of an operand and leaving the unwanted portions behind. This is accomplished by placing a mask in the *Q*-register consisting of 1's in the desired bit positions and 0's in the others. The logical product formed will contain those bits of the operand that correspond to the 1's in the *Q*-register. All other bits of the logical product will be 0's. Simply stated, in logical product operations the following rules apply:

$$\begin{aligned} Q_n &= 0\ 1\ 1\ 0 \\ Y_n &= \underline{1\ 0\ 1\ 0} \\ \text{nth bit of } L[Y(0)] &= 0\ 0\ 1\ 0 \end{aligned}$$

The following examples illustrates these rules:

$$\begin{aligned} (Q) &= 00000000000000000111000111000 \\ Y &= \underline{00000000000000000001010011100} \\ L[Y(Q)] &= 00000000000000000001000001100 \end{aligned}$$

SELECTIVE SET, (A) v Y

Selective set operations are used to force 1's into selected bit positions of the *A*-register. The bit positions that 1's are forced into are determined by the condition of the corresponding bit positions in the operand; that is, if either or both of these bit positions contain a 1, the result will be a 1. If both contain a 0, the result will be a 0. Simply stated in selective set operations the following rules apply:

$$\begin{aligned} A_n &= 0\ 1\ 1\ 0 \\ Y_n &= \underline{1\ 0\ 1\ 0} \\ \text{nth bit of } (A) \text{ v } Y &= 1\ 1\ 1\ 0 \end{aligned}$$

The following example illustrates these rules:

$$\begin{aligned} (A) &= 00000000000000000001010011100 \\ Y &= \underline{00000000000000000100100100100} \\ (A) \text{ v } Y &= 00000000000000000101110111100 \end{aligned}$$

SELECTIVE CLEAR. (A) ∧ N

Selective clear operations are used to force 0's into selected bit positions of the *A*-register. The bit positions that 0's are forced into are determined by the presence of 1's in the corresponding bit positions of the operand; that is, if a bit position in the operand contains a 1, the result will be a 0. Simply stated, the following rules apply in selective clear operations:

$$\begin{aligned} A_n &= 0\ 1\ 1\ 0 \\ Y_n &= 1\ 0\ 1\ 0 \\ \text{nth bit of } (A) \wedge Y &= 0\ 1\ 0\ 0 \end{aligned}$$

The following example illustrates these rules:

$$\begin{aligned} (A) &= 11111111111111111101110010111 \\ Y &= 00000000000000000100100100100 \\ (A) \wedge Y &= 11111111111111111001010010011 \end{aligned}$$

It should be noted that selective clear operations also perform a zero-masking function. If the above example is examined, this soon becomes apparent since in each case where a bit position of the operand contains a 0, the bit in the corresponding bit position in the *A*-register is lifted out and placed in the result.

SELECTIVE COMPLEMENT. (A) ⊕ Y

Selective complement operations are used to complement the bits in selected bit positions of the *A*-register. The bit positions that are to be complemented are determined by the presence of 1's in the corresponding bit positions of the operand; that is, if a bit position in the operand contains a 1, the bit in the corresponding bit position of the *A*-register will be complemented. Simply stated, the following rules apply in selective complement operations:

$$\begin{aligned} A_n &= 0\ 1\ 1\ 0 \\ Y_n &= 1\ 0\ 1\ 0 \\ \text{nth bit of } (A) \oplus Y &= 1\ 1\ 0\ 0 \end{aligned}$$

The following example illustrates these rules:

$$\begin{aligned} (A) &= 00000000000000000100101110111 \\ Y &= 00000000000000000111000111000 \\ (A) \oplus Y &= 00000000000000000011101001111 \end{aligned}$$

SELECTIVE SUBSTITUTE. L[(A)(Q)'] + L[Y(Q)]

Selective substitute operations are used for replacing bits in selected bit positions of the *A*-register with bits from the corresponding bit positions of an operand. The bits of the operand that will replace those in the *A*-register are specified by 1's in the *Q*-register. For example:

FIRST:

$$\begin{aligned} (Q)' &= 00000000000000000111000111000 \\ (A) &= 00000000000000000001101011110 \\ L[(A)(Q)'] &= 00000000000000000001000011000 \end{aligned}$$

SECOND:

$$\begin{aligned} (Q) &= 11111111111111111000111000111 \\ Y &= 00000000000000000110010101100 \\ L[Y(Q)] &= 00000000000000000010000100 \end{aligned}$$

FINALLY:

$$\begin{aligned} L[(A)(Q)'] &= 00000000000000000001000011000 \\ L[Y(Q)] &= 00000000000000000000010000100 \\ L[(A)(Q)'] + L[Y(Q)] &= 00000000000000000001010011100 \end{aligned}$$

ENTER LOGICAL PRODUCT

CLASS: Read

FUNCTION CODE: 40

MNEMONIC: ENT · *LP*

OPERATION: $L[Y(Q)] \longrightarrow A$

DESCRIPTION: This instruction forms the logical product of the contents of the *Q*-register and a 30-bit operand and retains it in the *A*-register.

k DESIGNATORS: The operand in the logical product operation is derived as follows:

k = 0: $L[\overline{y}(Q)] \longrightarrow A$. The lower half of the operand is $\overline{y}$ – the low-order 15 bits contained in the instruction word after *B*-register modification; the upper half is all 0's.

k = 1: $L[Y_L(Q)] \longrightarrow A$. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper half is all 0's.

k = 2: $L[Y_U(Q)] \rightarrow A$. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is all 0's.

k = 3: $L[Y(Q)] \rightarrow A$. The operand is the 30 bits contained in the storage location at address $\bar{y}$.

k = 4: $L[X\bar{y}(Q)] \rightarrow A$. The lower half of the operand is $\bar{y}$ – the lower-order 15 bits contained in the instruction word after *B*-register modification; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 of $\bar{y}$ is a 0, or it will be all 1's if bit 14 is a 1.

k = 5: $L[XY_L(Q)] \rightarrow A$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1.

k = 6: $L[XY_U(Q)] \rightarrow A$. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1.

k = 7: $L[(A)(Q)] \rightarrow A$. The operand is the 30 bits contained in the *A*-register.

i DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if $(A)_f$ contains an even number of 1's, all 1's, or all 0's.

j = 3: skip *NI* if $(A)_f$ contains an odd number of 1's.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is – or – 0.

ADD LOGICAL PRODUCT

CLASS: Read

FUNCTION CODE: 41

MNEMONIC: ADD. LP

OPERATION: $(A) + L[Y(Q)] \rightarrow A$

DESCRIPTION: This instruction adds the contents of the A-register to the logical product of the contents of the *Q*-register and a 30 bit operand and retains the sum that is formed in the *A*-register.

k DESIGNATORS: The operand in the logical product operation is derived as follows:

k = 0: $(A) + L[\bar{y}(Q)] \rightarrow A$. The lower half of the operand is $\bar{y}$ – the low-order 15 bits contained in the instruction word after *B*-register modification; the upper half is all 0's.

k = 1: $(A) + L[Y_L(Q)] \rightarrow A$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is all 0's.

k = 2: $(A) + L[Y_U(Q)] \rightarrow A$. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half in all 0's.

k = 3: $(A) + L[Y(Q)] \rightarrow A$. The operand is the 30 bits contained in the storage location at address $\bar{y}$.

k = 4: $(A) + L[X\bar{y}(Q)] \rightarrow A$. The lower- half of the operand is $\bar{y}$ – the low-order 15 bits contained in the instruction word after *B*-register modification; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 of $\bar{y}$ is a 0, or it will be all 1's if bit 14 is a 1.

k = 5: $(A) + L[XY_L(Q)] \rightarrow A$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1.

k = 6: $(A) + L[XY_U(Q)] \rightarrow A$. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper

half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1.

k = 7: $(A)+L[(A)(Q)] \longrightarrow A$. The operand is the 30 bits contained in the *A*-register.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if *(Q)* is + or + 0.

j = 3: skip *NI* if *(Q)* is − or − 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is − or − 0.

SUBTRACT LOGICAL PRODUCT

CLASS: Read

FUNCTION CODE: 42

MNEMONIC: *SUB · LP*

OPERATION: $(A) - L[Y(Q)] \longrightarrow A$

DESCRIPTION: This instruction subtracts the logical product of the contents of the *Q*-register and a 30-bit operand from the contents of the *A*-register and retains the difference that is formed in the *A*-register.

k DESIGNATORS: The operand in the logical product operation is derived as follows:

k = 0: $(A) - L[\overline{y}(Q)] \longrightarrow A$. The lower half of the operand is $\overline{y}$ − the low-order 15 bits contained in the instruction word after *B*-register modification; the upper-half is all 0's.

k = 1: $(A) - L[Y_L(Q)] \longrightarrow A$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\overline{y}$; the upper-half is all 0's.

k = 2: $(A) - L[Y_U(Q)] \longrightarrow A$. The lower half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper-half is all 0's.

k = 3: $(A) - L[Y(Q)] \longrightarrow A$. The operand is the 30-bits contained in the storage location at address $\overline{y}$.

k = 4: $(A) - L[X\overline{y}(Q)] \longrightarrow A$. The lower half of the operand is $\overline{y}$ − the low-order 15 bits contained in the instruction word after *B*-register modification; the upper half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 of *y* is a 0, or it will be all 1's if bit 14 is a 1.

k = 5: $(A) - L[XY_L(Q)] \longrightarrow A$. The lower half of the operand is the low-order 15 bits contained in the storage location at address $\overline{y}$, the upper half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1.

k = 6: $(A) - L[XY_U(Q)] \longrightarrow A$. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$, the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1.

k = 7: $(A) - L[(A)(Q)] \longrightarrow A$. The operand is the 30 bits contained in the *A*-register.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if *(Q)* is + or + 0.

j = 3: skip *NI* if *(Q)* is − or − 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is − or − 0.

SELECTIVE SET

CLASS: Read

FUNCTION CODE: 50

MNEMONIC: *SEL · SET*

OPERATION: $A \vee Y \longrightarrow A$; i.e., Set *(A) n* for *Y n = 1*.

DESCRIPTION: This instruction forces 1's into selected bit positions of the *A*-register. The bit positions that 1's are forced into are determined by the condition of the corresponding bit positions in the operand; that is, if either or both of these bit positions contain a 1, the result will be a 1. If both contain a 0, the result will be a 0.

k DESIGNATORS: The operand is derived as follows:

k = 0: $(A)\ v\ \bar{y} \longrightarrow A$. The lower half of the operand is $\bar{y}$ – the low-order 15 bits contained in the instruction word after *B*-register modification; the upper-half is all 0's.

k = 1: $(A)\ v\ Y_L \longrightarrow A$. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is all 0's.

k = 2: $(A)\ v\ Y_U \longrightarrow A$. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is all 0's.

k = 3: $(A)\ v\ Y \longrightarrow A$. The operand is the 30 bits contained in the storage location at address $\bar{y}$.

k = 4: $(A)\ v\ X\bar{y} \longrightarrow A$. The lower-half of the operand is $\bar{y}$ – the low-order 15 bits contained in the instruction word after *B*-register modification; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 of $\bar{y}$ is a 0, or it will be all 1's if bit 14 is a 1.

k = 5: $(A)\ v\ XY_L \longrightarrow A$. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1.

k = 6: $(A)\ v\ XY_U \longrightarrow A$. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1.

k = 7: not used.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if *(Q)* is + or + 0.

j = 3: skip *NI* if *(Q)* is – or – 0.

j = 4: skip *NI* if $(A)_f$ if + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is – or – 0.

SELECTIVE COMPLEMENT

CLASS: Read

FUNCTION CODE: 51

MNEMONIC: SEL · CP

OPERATION: $(A) \oplus Y \longrightarrow A$; i.e., complement $(A)n$ for $Yn = 1$.

DESCRIPTION: This instruction complements the bits in selected bit positions of the *A*-register. The bit positions that are to be complemented are determined by the presence of 1's in the corresponding bits positions of the operand; that is, if a bit position in the operand contains a 1, the bit in the corresponding bit position of the A-register will be complemented.

k DESIGNATORS: The operand is derived as follows:

k = 0: $(A) \oplus \bar{y} \longrightarrow A$. The lower-half of the operand is $\bar{y}$ – the low-order 15 bits contained in the instruction word after *B*-register modification; the upper half is all 0's.

k = 1: $(A) \oplus Y_L \longrightarrow A$. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is all 0's.

k = 2: $(A) \oplus Y_U \longrightarrow A$. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is all 0's.

k = 3: $(A) \oplus Y \longrightarrow A$. The operand is the 30-bits contained in the storage location at address $\bar{y}$.

k = 4: $(A) \oplus X\bar{y} \longrightarrow A$. The lower-half of the operand is $\bar{y}$ – the low-order 15 bits contained in the instruction word after *B*-register modification; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 of $\bar{y}$ is a 0, or it will be all 1's if bit 14 is a 1.

k = 5: $(A) \oplus XY_L \longrightarrow A$. The lower-half of the the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1.

k = 6: $(A) \oplus XY_U \longrightarrow A$. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$, the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's bit 29 is a 1.

k = 7: $(A) \oplus (A) \longrightarrow A$. The operand is the 30 bits contained in the *A*-register. $(A)_f$ will contain +0 when this value for *k* is used.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if *(Q)* is + or + 0.

j = 3: skip *NI* if *(Q)* is – or – 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)f$ is – or – 0.

SELECTIVE CLEAR

CLASS:	Read
FUNCTION CODE:	52
MNEMONIC:	SEL · CL
OPERATION:	$(A) \wedge Y \longrightarrow A$; i.e., clear $(A)n$ for $Yn = 1$.

DESCRIPTION: This instruction forces 0's into selected bit position of the *A*-register. The bit positions that 0's are forced into are determined by the presence of 1's in the corresponding bit positions of the operand.

k DESIGNATORS: The operand is derived as follows:

k = 0: $(A) \wedge \bar{y} \longrightarrow A$. The lower-half of the operand is $\bar{y}$ – the low-order 15 bits contained in the instruction word after *B*-register modification, the upper-half is all 0's.

k = 1: $(A) \wedge Y \longrightarrow A$. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is all 0's.

k = 2: $(A) \wedge Y_U \longrightarrow A$. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is all 0's.

k = 3: $(A) \wedge Y \longrightarrow A$. The operand is the 30 bits contained in the storage location at address $\bar{y}$.

k = 4: $(A) \wedge X\bar{y} \longrightarrow A$. The lower-half of the operand is $\bar{y}$ – the low-order 15 bits contained in the instruction word after *B*-register modification; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 of $\bar{y}$ is 0, or it will be all 1's if bit 14 is a 1.

k = 5: $(A) \wedge XY \longrightarrow A$. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1.

k = 6. $(A) \wedge XY \longrightarrow A$. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1.

k = 7: not used.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if *(Q)* is + or + 0.

j = 3: skip *NI* if *(Q)* is − or − 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is − or − 0.

SELECTIVE SUBSTITUTE

CLASS: Read

FUNCTION CODE: 53

MNEMONIC: SEL · SU

OPERATION: L[(A)(Q)'] + L[Y(Q)]⟶*A*; i.e., $Yn \rightarrow (A)n$ for $(Q)n = 1$.

DESCRIPTION: This instruction replaces bits in selected bit positions of the *A*-register with bits from the corresponding bit positions of an operand. The bits of the operand that will replace those in the *A*-register are specified by 1's in register Q.

k DESIGNATORS: The operand that contains the bits that will replace the bits in the selected bit positions of the *A*-register is derived as follows:

k = 0: $L[(A)(Q)'] + L[\bar{y}(Q)] \rightarrow A$. The lower-half of the operand is $\bar{y}$ − the low-order 15 bits contained in the instruction word after *B*-register modification; the upper-half is all 0's.

k = 1: $L[(A)(Q)'] + L[Y_L\ (Q)] \rightarrow A$. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is all 0's.

k = 2: $L[(A)(Q)'] + L[Y_U\ (Q)] \rightarrow A$. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is all 0's.

k = 3: $L[(A)(Q)'] + L[Y(Q)] \rightarrow A$. The operand is the 30 bits contained in the storage location at address $\bar{y}$.

k = 4: $L[(A)(Q)'] + L[X\bar{y}(Q)] \rightarrow A$. The lower-half of the operand is $\bar{y}$ − the low-order 15 bits contained in the instruction word after *B*-register modification; the upper-half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 14 of *y* is a 0, or it will be all 1's if bit 14 is a 1.

k = 5: $L[(A)(Q)'] + L[XY_L\ (Q)] \rightarrow A$. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1.

k = 6: $L[(A)(Q)'] + L[XY_U\ (Q)] \rightarrow A$. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1.

k = 7: not used.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip

j = 1: skip *NI*.

j = 2: skip *NI* if *(Q)* is + or + 0.

j = 3: skip *NI* if *(Q) is − or − 0.*

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is − or − 0.

REPLACE LOGICAL PRODUCT

CLASS: ~~Read~~ REPLACE

FUNCTION CODE: 44

MNEMONIC: RPL · LP

OPERATION: $L[Y(Q)] \rightarrow Y$ and *A*

DESCRIPTION: This instruction forms the logical product of the contents of the *Q*-register and a 30 bit operand, retains the logical product in the A-register, and stores this logical product in the storage location that the operand was obtained form.

k DESIGNATORS: The operand in the logical product operation and the logical product that is stored in the storage location that the operand was obtained from are derived as follows:

k = 0, 4, or 7: not used.

k = 1; $L[Y_L(Q)] \rightarrow A$, $(A)_L \rightarrow Y_L$; Y_U is undisturbed. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is all 0's. After the logical product is formed in the *A*-register, the low-order 15 bits of this product are stored in the lower-half of the storage location from which the operand was obtained; the upper-half of the storage location is undisturbed.

k = 2: $L[Y_U(Q)] \rightarrow A$, $(A)_L \rightarrow Y_U$; Y_L is undisturbed. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is all 0's. After the logical product is formed in the *A*-register, the low-order 15 bits of this product are stored in the upper-half of the storage location from which the operand was obtained; the lower-half of the storage location is undisturbed.

k = 3: $L[Y(Q)] \rightarrow A, Y$. The operand is the 30 bits contained in the storage location at address $\bar{y}$. After the logical product is formed in the *A*-register, it is stored in the storage location from which the operand was obtained.

k = 5: $L[XY_L(Q)] \rightarrow A$, $(A)_L \rightarrow Y_L$; Y_U is undisturbed. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1. After the logical product is formed in the *A*-register, the low-order 15 bits of this product are stored in lower-half of the storage location from which the operand was obtained; the upper-half of the storage location is undisturbed.

k = 6; $L[XY_U(Q)] \rightarrow A$, $(A)_L \rightarrow Y_U$; Y_L is undisturbed. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1. After the logical product is formed in the *A*-register, the low-order 15 bits of this product are stored in the upper-half of the storage location from which the operand was obtained; the lower-half of the storage location is undisturbed.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if $(A)_f$ contains an even number of 1's, all 1's or all 0's.

j = 3: skip *NI* if $(A)_f$ contains an odd number of 1's.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is − or − 0.

REPLACE A + LOGICAL PRODUCT

CLASS: Replace

FUNCTION CODE: 45

MNEMONIC: RPL · A + LP

OPERATION: $(A) + L[Y(Q)] \rightarrow Y$ and *A*.

DESCRIPTION: This instruction forms the logical product of the contents of the *Q*-register and a 30-bits operand, adds this product to the contents of the *A*-register, retains the sum that is formed in the *A*-register, and stores this sum in the storage location from which the operand was obtained.

k DESIGNATORS: The operand in the logical product operation and the sum that is stored in the storage location from which the operand was obtained are derived as follows:

k = 0, 4, or 7: not used.

k = 1: $(A) + L[Y_L(Q)] \rightarrow A$, $(A)_L \rightarrow Y_L$; Y_U is undisturbed. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is all 0's. After the sum is formed in the *A*-register, the low-order 15 bits of this sum are stored in the in the lower-half of the storage location from which the operand was obtained; the upper-half of the storage location is undisturbed.

k = 2: $(A)+L[Y_U(Q)] \rightarrow A$, $(A)_L \rightarrow Y_U$; Y_L is undisturbed. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is all 0's. After the sum is formed is the A-register, the low-order 15 bits of this sum are stored in the upper-half of the storage location from which the operand was obtained; the lower-half of the storage location is undisturbed.

k = 3: $(A) + L[Y(Q)] \rightarrow A, Y$. The operand is the 30 bits contained in the storage location at address $\bar{y}$. After the sum is formed in the *A*-register, it is stored in the storage location from which the operand was obtained.

k = 5: $(A)+L[XY_L(Q)] \rightarrow A$, $(A)_L \rightarrow Y_L$; Y_U is undisturbed. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1. After the sum is formed in the *A*-register, the low-order 15 bits of this sum are stored in the lower-half of the storage location from which the operand was obtained; the upper-half of the storage location is undisturbed.

k = 6: $(A)+L[XY_U(Q)] \rightarrow A$, $(A)_L \rightarrow Y_U$; Y_L is undisturbed. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1. After the sum is formed in the *A*-register, the low-order 15 bits of this sum are stored in the upper-half of the storage location from which the operand was obtained; the lower-half of the storage location is undisturbed.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if (Q) is + or + 0.

j = 3: skip *NI* if (Q) is − or − 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6. skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is − or − 0.

REPLACE A − LOGICAL PRODUCT

CLASS:	Replace
FUNCTION CODE:	46
MNEMONIC:	RPL • A − LP
OPERATION:	$(A) - L[Y(Q)] \rightarrow Y$ *and* A.

DESCRIPTION: This instruction forms the logical product of the contents of the *Q*-register and a 30-bit operand, subtracts this product from the contents of the *A*-register, retains the difference in the *A*-register, and stores it in the storage location from which the operand was obtained.

k DESIGNATORS: The operand in the logical product operation and the difference stored in the storage location from which the operand was obtained are derived as follows:

k = 0: 4; or 7: not used.

k = 1: $(A)-L[Y_L(Q)] \rightarrow A$, $(A)_L \rightarrow Y_L$; Y_U is undisturbed. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is all 0's. After the difference is formed in the *A*-register, the low-order 15 bits are stored in the lower-half of the storage location from which the operand was obtained; the upper-half of the storage location is undisturbed.

k = 2: $(A) - L[Y_U(Q)] \rightarrow A$, $(A)_L \rightarrow Y_U$; Y_L is undisturbed. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper half is all 0's. After the difference is formed in the *A*-register, the low-order 15 bits are stored in the upper-half of the storage location from which the operand was obtained; the lower-half of the storage location is undisturbed.

k = 3: $(A) - L[Y(Q)] \rightarrow A, Y$. The operand is the 30 bits contained in the storage location at address $\bar{y}$. After the difference is formed in the *A*-register, it is stored in the storage location from which the operand was obtained.

k = 5: $(A) - L[XY_L(Q)] \rightarrow A$, $(A)_L \rightarrow Y_L$; Y_U is undisturbed. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1. After the difference is formed in the *A*-register, the low-order 15 bits are stored in the lower-half of the storage location from which the operand was obtained; the upper-half of the storage location is undisturbed.

k = 6: $(A) - L[XY_U(Q)] \rightarrow A$, $(A)_L \rightarrow Y_U$; Y_L is undisturbed. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1. After the difference is formed in the *A*-register, the low-order 15 bits are stored in the upper-half of the storage location from which the operand was obtained; the lower-half of the storage location is undisturbed.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if (Q) is + or + 0.

j = 3: skip *NI* if (Q) is − or − 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is − or − 0.

STORE LOGICAL PRODUCT

CLASS: Store

FUNCTION CODE: 47

MNEMONIC: STR • LP

OPERATION: $L[(A)(Q)] \rightarrow Y$

DESCRIPTION: This instruction forms the logical product of the contents of the *Q*-register and the *A*-register and stores this product in a storage location.

k DESIGNATORS: The product that is stored in the storage location is derived as follows:

k = 0: $L[(A)(Q)] \rightarrow Q$. The logical product is stored in the *Q*-register. With the exception of this value for *k*, the *Q*-register is undisturbed by this instruction.

k = 1: $L[(A)(Q)] \rightarrow Y_L$; Y_U is undisturbed. The low-order 15 bits of the logical product are stored in the lower-half of the storage location at address $\bar{y}$; the upper-half of the storage location is undisturbed.

k = 2: $L[(A)(Q)] \rightarrow Y_U$; Y_L is undisturbed. The low-order 15 bits of the logical product are stored in the upper-half of the storage location or address $\bar{y}$; the lower-half of the storage location is undisturbed.

k = 3: $L[(A)(Q)] \rightarrow Y$. The logical product is stored in the storage location at address $\bar{y}$.

k = 4: $L[(A)(Q)] \rightarrow A$. The logical product is stored in the *A*-register. With the exception of this value for *k*, the *A*-register is undisturbed by this instruction.

k = 5: The complement of $L[(A)(Q)] \rightarrow Y_L$; Y_U is undisturbed. The low-order 15 bits of the complement of the logical product are stored in the lower-half of the storage location at address $\bar{y}$; the upper-half of the storage location is undisturbed.

k = 6: The complement of $L[(A)(Q)] \rightarrow Y_U$; Y_L is undisturbed. The complement of the low-order 15 bits of the logical product are stored in the upper-half of the storage location at address $\bar{y}$; the lower-half of the storage location is undisturbed.

k = 7

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if $(Q)_f$ is + or + 0.

j = 3: skip *NI* if $(Q)_f$ is − or − 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ *is + or + 0.*

j = 7: skip *NI* if $(A)_f$ is – or – 0.

REPLACE SELECTIVE SET

CLASS:	Replace
FUNCTION CODE:	54
MNEMONIC:	RSE • SET
OPERATION:	$(A)\ v\ Y \rightarrow Y$ and A; i.e., set $(A)n$ for $Yn = 1 \rightarrow Y$ and A.

DESCRIPTION: This instruction forces 1's into selected bit positions of the *A*-register. The bit positions that 1's are forced into are determined by the presence of 1's in the corresponding bit positions of the operand. After the selective set operation is performed, the result is retained in the *A*-register and stored in the storage location from which the operand was obtained.

k DESIGNATORS: The operand in the selective set operation and the result that is stored in the storage location from which the operand was obtained are derived as follows:

k = 0, 4, or 7: not used.

k = 1: $(A)\ v\ Y_L \rightarrow A$, $(A)_L \rightarrow Y_L$; Y_U is undisturbed. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is all 0's. After the selective set operation is performed, the low-order 15 bits of the result are stored in the lower-half of the storage location from which the operand was obtained; the upper-half of the storage location is undisturbed.

k = 2: $(A)\ v\ Y_U \rightarrow A$, $(A)_L \rightarrow Y_U$; Y_L is undisturbed. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is all 0's. After the selective set operation is performed, the low-order 15 bits of the result are stored in the upper-half of the storage location from which the operand was obtained; the lower-half of the storage location is undisturbed.

k = 3: $(A)\ v\ Y \rightarrow A$, Y. The operand is the 30 bits contained in the storage location at address $\bar{y}$. After the selective set operation is performed, the result is stored in the storage location from which the operand was obtained from.

k = 5: $(A)\ v\ Y_L \rightarrow A$, $(A)_L \rightarrow Y_L$; Y_U is undisturbed. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1. After the selective set operation is performed, the low-order 15 bits of the result are stored in the lower-half of the storage location from which the operand was obtained from; the upper-half of the storage location is undisturbed.

k = 6: $(A)\ v\ XY_U \rightarrow A$, $(A)_L \rightarrow Y_U$; Y_L is undisturbed. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1. After the selective set operation is performed, the low-order 15 bits of the result are stored in the upper-half of the storage location from which the operand was obtained, the lower-half of the storage location is undisturbed.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if *(Q)* is + or + 0.

j = 3: skip *NI* if *(Q)* is – or – 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is – or – 0.

REPLACE SELECTIVE COMPLEMENT

CLASS:	Replace
FUNCTION CODE:	55
MNEMONIC:	RSE • CP
OPERATION:	$(A) \oplus Y \rightarrow Y$ and A; i.e., complement $(A)n$ for $Yn = 1 \rightarrow Y$ and A.

DESCRIPTION: This instruction complements the bits in selected bit positions of the *A*-register. The bits complemented are determined by the presence of 1's in the corresponding bit positions of the operand. After the selective complement operation is performed, the result is retained in the *A*-register and stored in the storage location from which the operand was obtained.

k DESIGNATORS: The operand in the selective complement operation and the result stored in the storage location from which the operand was obtained are derived as follows:

k = 0, 4, or 7: not used.

k = 1: $(A) \oplus Y_L \rightarrow A, (A)_L \rightarrow Y_L$; Y_U is undisturbed. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is all 0's. After the selective complement operation is performed, the low-order 15 bits of the result are stored in the lower-half of the storage location from which the operand was obtained; the upper-half of the storage location is undisturbed.

k = 2: $(A) \oplus Y_U \rightarrow A, (A)_L \rightarrow Y_U$; Y_L is undisturbed. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is all 0's. After the selective complement operation is performed, the low-order 15 bits of the result are stored in the upper-half of the storage location from which the operand was obtained; the lower-half of the storage location is undisturbed.

k = 3: $(A) \oplus Y \rightarrow A, Y$. The operand is the 30 bits contained in the storage location at address $\bar{y}$. After the selective complement operation is performed, the result is stored in the storage location from which the operand was obtained.

k = 5: $(A) \oplus XY_L \rightarrow A, (A)_L \rightarrow Y_L$; Y_U is undisturbed. The lower-half of the operand is the low-order 15 bits contained in the storage location at address *y*. The upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1. After the selective complement operation is performed, the low-order 15 bits of the result are stored in the lower-half of the storage location from which the operand was obtained; the upper-half of the storage location is undisturbed.

k = 6: $(A) \oplus XY_U \rightarrow A, (A)_L \rightarrow Y_U$; Y_L is undisturbed. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1. After the selective complement operation is performed, the low-order 15 bits of the result are stored in the upper-half of the storage location from the operand was obtained, the lower-half of the storage location is undisturbed.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if *(Q)* is + or + 0.

j = 3: skip *NI* if *(Q)* is − or − 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is − or − 0.

REPLACE SELECTIVE CLEAR

CLASS:	Replace
FUNCTION CODE:	56
MNEMONIC:	RSE • CL
OPERATION:	$(A) \wedge Y \rightarrow Y$ and *A*; i. e., clear $(A)n$ for $Yn = 1 \rightarrow Y$ and *A*.

DESCRIPTION: This instruction forces 0's into selected bit positions of the *A*-register. The bit positions that 0's are forced into are determined by the presence of 1's in the corresponding bit positions of the operand. After the selective clear operation is performed, the result is retained in the *A*-register and stored in the storage location from which the operand was obtained.

k DESIGNATORS: The operand in the selective clear operation and the result that is stored in the storage location from which the operand was obtained are derived as follows:

k = 0, 4, or 7: not used.

k = 1: $(A) \wedge Y_L \rightarrow A, (A)_L \rightarrow Y_L$; Y_U is undisturbed. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is all 0's. After the selective clear operation is performed, the low-order 15 bits of the result are stored in the lower-half of the storage location from which the operand was obtained; the upper-half of the storage location is undisturbed.

k = 2: $(A) \wedge Y_U \rightarrow A, (A)_L \rightarrow Y_U$; Y_L is undisturbed. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is all 0's. After the selective clear operation is performed, the low-order 15 bits of the result are stored in the upper-half of the storage location from which the operand was obtained; the lower-half of the storage location is undisturbed.

k = 3: $(A) \wedge Y \longrightarrow A, Y$. The operand is the 30 bits contained in the storage location is address $\bar{y}$. After the selective clear operation is performed, the result is stored in the storage location from which the operand was obtained.

k = 5: $(A) \wedge XY_L \rightarrow A, (A)_L \rightarrow Y_L$; Y_U is undisturbed. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1. After the selective clear operation is performed, the low-order 15 bits of the result are stored in the lower-half of the storage location from which the operand was obtained; the upper-half of the storage location is undisturbed.

k = 6: $(A) \wedge XY_U \rightarrow A, (A)_L \rightarrow Y_U$; Y_L is undisturbed. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1. After the selective clear operation is performed, the low-order 15 bits of the result are stored in the upper-half of the storage location from which the operand was obtained from; the lower-half of the storage location is undisturbed.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if (Q) is + or + 0.

j = 3: skip *NI* if (Q) is − or − 0.

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is $\bar{+}$ or $\bar{+}$ 0.

REPLACE SELECTIVE SUBSTITUTE

CLASS: Replace

FUNCTION CODE: 57

MNEMONIC: RSE • SU

OPERATION: $L[(A)(Q)'] + L[Y(Q)] \rightarrow Y$ *and* A; i.e., $Yn \rightarrow (A)n$ for $(Q)n = 1 \rightarrow Y$ and A.

DESCRIPTION: This instruction replaces bits in the selected bit positions of the *A*-register with the bits in the corresponding bit positions of the operand. The bits that are to replace those in the *A*-register are determined by the presence of 1's in the *Q*-register. After the selective substitute operation is performed, the result is retained in the *A*-register and stored in the storage location from which the operand was obtained.

k DESIGNATORS: The operand in the selective substitute operation and the result that is stored in the storage location from which the operand was obtained are derived as follows:

k = 0, 4, or 7: not used.

k = 1: $L[(A)(Q)'] + L[Y_L(Q)] \rightarrow A, (A)_L \rightarrow Y_L$; Y_U is undisturbed. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is all 0's. After the selective substitute operation is performed, the low-order 15 bits of the result are stored in the lower-half of the storage location from which the operand was obtained; the upper-half of the storage location is undisturbed.

k = 2: $L[(A)(Q)'] + L[Y_U(Q)] \rightarrow A, (A)_L \rightarrow Y_U$: Y_L is undisturbed. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half

is all 0's. After the selective substitute operation is performed, the low-order 15 bits of the result are stored in the upper-half of the storage location from which the operand was obtained from; the lower-half of the storage location is undisturbed.

k = 3: $L[(A)(Q)'] + L[Y(Q)] \rightarrow A$, Y. The operand is the 30 bits contained in the storage location at address $\bar{y}$. After the selective substitute operation is performed, the result is stored in the storage location from which the operand was obtained.

k = 5: $L[(A)(Q)'] + L[XY_L(Q)] \rightarrow A$, $(A)_L \rightarrow Y_L$; Y_U is undisturbed. The lower-half of the operand is the low-order 15 bits contained in the storage location at address y; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1. After the selective substitute operation is performed, the low-order 15 bits of the result are stored in the lower-half of the storage location that the operand was obtained from; the upper-half of the storage location is undisturbed.

k = 6: $L[(A)(Q)'] + L[XY_U(Q)] \rightarrow A$, $(A)_L \rightarrow Y_U$; Y_L is undisturbed. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1. After the selective substitute operation is performed, the low-order 15 bits of the result are stored in the upper-half of the storage location from which the operand was obtained; the lower-half of the storage location is undisturbed.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if *(Q)* is + or + 0.

j = 3: skip *NI* if *(Q)* is − or − 0

j = 4: skip *NI* if $(A)_f$ is + 0.

j = 5: skip *NI* if $(A)_f$ is not + 0.

j = 6: skip *NI* if $(A)_f$ is + or + 0.

j = 7: skip *NI* if $(A)_f$ is − or − 0.

COMPARE MASKED

CLASS: Read

FUNCTION CODE: 43

MNEMONIC: COM • MASK

OPERATION: $(A) - L[Y(Q)] = D$; test D to determine skip.

DESCRIPTION: This instruction compares the contents of the *A*-register to a masked operand. The comparison is made by forming the logical product of the *Q*-register and a specified operand, subtracting this logical product from the contents of the *A*-register, and then examining *D*, the difference that is formed. The contents of the *A*-register are undisturbed by this instruction.

k DESIGNATORS: The operand in the logical product operation is derived as follows:

k = 0: $(A) - L[\bar{y}(Q)] = D$. The lower-half of the operand is $\bar{y}$ − the low-order 15 bits contained in the instruction word after *B*-register modification; the upper-half is all 0's.

k = 1: $(A) - L[Y_L(Q)] = D$. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$, the upper-half is all 0's.

k = 2: $(A) - L[Y_U(Q)] = D$. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is all 0's.

k = 3: $(A) - L[Y(Q)] = D$. The operand is the 30 bits contained in the storage location at address $\bar{y}$.

k = 4: $(A) - L[X\bar{y}(Q)] = D$. The lower-half of the operand is $\bar{y}$, the low-order 15 bits contained in the instruction word after B-register modification; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 of $\bar{y}$ is a 0, or it will be all 1's if bit 14 is a 1.

k = 5: $(A) - L[XY_L(Q)] = D$. The lower-half of the operand is the low-order 15 bits contained in the storage location at address $\bar{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 14 in the storage location is a 0, or it will be all 1's if bit 14 is a 1.

k = 6: $(A) - L[XY_U(Q)] = D$. The lower-half of the operand is the high-order 15 bits contained in the storage location at address $\overline{y}$; the upper-half is an extension of the sign bit. The upper-half of the operand will be all 0's if bit 29 in the storage location is a 0, or it will be all 1's if bit 29 is a 1.

k = 7: $(A) - L[(A)(Q)] = D$. The operand is the 30 bits contained in the *A*-register.

j DESIGNATORS: The skip conditions are determined as follows:

j = 0: no skip.

j = 1: skip *NI*.

j = 2: skip *NI* if *(Q)* is + or + 0.

j = 3: skip *NI* if *(Q)* is − or − 0.

j = 4: skip *NI* if *D* is + 0.

j = 5: skip *NI* if *D* is not + 0.

j = 6: skip *NI* if *D* is + or + 0.

j = 7: skip *NI* if *D* is − or − 0.

7. MODIFYING INSTRUCTIONS

Modifying instructions cause the program to take one of three actions:

- Execute the following instruction the number of times specified.
- Skip the next instruction and continue in sequence.
- Jump to another portion of the program.

REPEAT

CLASS: Read

FUNCTION CODE: 70

MNEMONIC: RPT·

OPERATION: Repeat *NI Y* times

DESCRIPTION: This instruction repeats the instruction immediately following it Y times. The repeat counter, Y, is a 15 bit positive number where $0 \leq Y \leq 32767_{10}$. Y is determined by the k-designator in the Repeat instruction. After k interpretation is made, Y is stored in B-register 7. If $Y = 0$, the instruction immediately following the Repeat instruction is skipped. If $Y \neq 0$, the repeat mode, as specified by the j-designator of the Repeat instruction, is initiated. It should be noted that during the repeat mode, modification of $\overline{y}$ takes place in the U-register; therefore, the repeated instruction, as stored in the Computer, is not altered. All interrupts are locked out during the repeat mode.

k DESIGNATORS: The repeat counter, Y, is derived as follows:

k = 0 or 4: $Y = \overline{y}$. The repeat counter is $\overline{y}$, the low-order 15 bits contained in the instruction word after B-register modification.

k = 1, 3, or 5: $Y = Y_L$. The repeat counter in the low-order 15 bits contained in the storage loca- at address $\overline{y}$.

k = 2 or 6: $Y = Y_U$. The repeat counter is the high-order 15 bits contained in the storage at address $\overline{y}$.

k = 7: $Y = A_L$. The repeat counter is the low-order 15 bits contained in the A-register.

j DESIGNATORS: The repeat mode is determined as follows:

j = 0 or 4: If the repeated instruction is a replace instruction, the result is replaced at address $\overline{y} + (B)_6$ rather than $\overline{y}$. For all other repeated instructions, this value for j does not cause any modification.

j = 1 or 5: If the repeated instruction is a replace instruction, the result is replaced at address $\overline{y} + (B)_6$ rather than $\overline{y}$. For all repeated instructions, including replace instructions, $\overline{y}$ is increased by 1 following each execution.

j = 2 or 6: If the repeated instruction is a replace instruction, the result is replaced at address $\overline{y} + (B)_6$ rather than $\overline{y}$. For all repeated instructions, including replace instructions, $\overline{y}$ is decreased by 1 following each execution.

j = 3 or 7: If the repeated instruction is a replace instruction, the result is replaced at address $\overline{y} + (B)_6$ rather than $\overline{y}$. For all repeated instructions, including replace instructions, the initial B-register modification specified in the instruction is performed prior to each execution; that is, for the Mth execution of the repeated instruction, $\overline{y} = y + M \times (B)j$.

The repeat mode is terminated if one of the following conditions occurs:

1. The repeated instruction has been executed Y times. In this case the repeat mode is terminated with $(B)_7 = 0$.

2. If a skip occurs at the M*th* execution of the repeated instruction. In this case the repeat mode is terminated with $(B)_7 = Y - M$.

3. If a jump instruction is repeated, the *P*-register will be set Y times. After the final execution, the jump will occur and the repeat mode will be terminated with $(B)_7 = 0$.

B SKIP ON B_j

CLASS: Read

FUNCTION CODE: 71

MNEMONIC: BSK • Bn

OPERATION: Skip *NI* if $(B)j = Y$. If $(B)j \neq Y$, add one to $(B)j$ and execute *NI*.

DESCRIPTION: This instruction compares the contents of the specified *B*-register with a 15-bit operand. If the contents of the *B*-register are not equal to the operand, 1 is added to the contents of the *B*-register and the program executes the next instruction. If they are equal, the *B*-register is cleared to 0 and the next instruction is skipped.

k DESIGNATORS: The operand is derived as follows:

k = 0 or 4: $Y = \overline{y}$. The operand is the low-order 15 bits contained in the instruction word after *B*-register modification.

k = 1, 3, or 5: $Y = Y_L$. The operand is the low-order 15 bits contained in the storage location at address $\overline{y}$.

k = 2 or 6: $Y = Y_U$. The operand is the high-order 15 bits contained in the storage location at address $\overline{y}$.

k = 7: $Y = A_L$. The operand is the low-order 15 bits contained in the *A*-register.

j DESIGNATORS: The *B*-register is specified as follows:

j = 0: *B*-register 0, skip *NI* if $Y = +0$.

j = 1: *B*-register 1.

j = 2: *B*-register 2.

j = 3: *B*-register 3.

j = 4: *B*-register 4.

j = 5: *B*-register 5.

j = 6: *B*-register 6.

j = 7: *B*-register 7.

B JUMP ON Bj

CLASS: Read

FUNCTION CODE: 72

MNEMONIC: BJP • Bn

OPERATION: If $(B)j \neq 0$, subtract 1 from $(B)j$ and jump to address Y. If $(B)j = 0$, execute *NI*.

DESCRIPTION: This instruction examines the contents of the specified *B*-register to determine whether or not they are equal to 0. If the specified *B*-register contains a 0 or if j equals 0, execute the next instruction. If j is unequal to 0, 1 is subtracted from the contents of the *B*-register and a jump to address Y is made.

k DESIGNATORS: The address to which the jump is made, Y, is derived as follows:

k = 0 or 4: $Y = \overline{y}$. The address is $\overline{y}$, the low-order 15 bits contained in the instruction word after *B*-register modification.

k = 1, 3, or 5: $Y = Y_L$. The address is the low-order 15 bits contained in the storage location at address $\overline{y}$.

k = 2 or 6: $Y = Y_U$. The address is the high-order 15 bits contained in the storage location at address $\overline{y}$.

k = 7: $Y = A_L$. The address is the low-order 15 bits contained in the *A*-register.

j DESIGNATORS: The *B*-register is specified as follows:

j = 0: *B*-register 0.

j = 1: *B*-register 1.

j = 2: *B*-register 2.

j = 3: *B*-register 3.

j = 4: *B*-register 4.

j = 5: *B*-register 5.

j = 6: *B*-register 6.

j = 7: *B*-register 7.

8. JUMP INSTRUCTIONS

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

JUMP (arithmetic)

CLASS: Read

FUNCTION CODE: 60

MNEMONIC: JP•,RIL • , RILJP•

OPERATION: Jump to address *Y* if the jump condition is satisfied. If the jump condition is not satisfied, execute *NI*.

DESCRIPTION: This instruction transfers program control to another section of the program depending upon the condition of the contents of either the *A*- or *Q*-register. If a jump condition is satisfied, a jump is made to address *Y*. If the condition is not satisfied, the instruction immediately following the jump instruction is executed.

k DESIGNATORS: The address to which the jump is made, *Y*, is derived as follows:

k = 0, or 4: $Y = \bar{y}$. The address is $\bar{y}$ – the low-order 15 bits contained in the instruction word after *B*-register modification.

k = 1, 3, or 5: $Y = Y_L$. The address is the low-order 15 bits contained in the storage location at address $\bar{y}$.

k = 2 or 6: $Y = Y_U$. The address is the high-order 15 bits contained in the storage location at address $\bar{y}$.

k = 7: $Y = A_L$. The address is the low-order 15 bits contained in the *A*-register.

j DESIGNATORS: The jump conditions are determined as follows:

j = 0: Do not jump; clear bootstrap and interrupt modes.

j = 1: Execute jump; clear bootstrap and interrupt modes.

j = 2: Execute jump if *(Q)* is + or + 0.

j = 3: Execute jump if *(Q)* is – or – 0.

j = 4: Execute jump if *(A)* is + 0.

j = 5: Execute jump if *(A)* is not + 0.

j = 6: Execute jump if *(A)* is + or + 0.

j = 7: Execute jump if *(A)* is – or – 0.

JUMP (manual)

CLASS: Read

FUNCTION CODE: 61

MNEMONIC: JP•

OPERATION: Jump to address Y if the jump condition is satisfied. If the jump condition is not satisfied, execute *NI*.

DESCRIPTION: This instruction transfers program control to another section of the program as directed by the key setting on the maintenance panel. If a jump condition is satisfied, a jump will be made to address Y. In certain cases the key setting will cause a jump and stop operation if the jump condition is satisfied. In these situations, the Computer stops with the P-register set to execute the instruction stored at address Y. In all cases where the jump condition is not satisfied, the instruction immediately following the jump instruction is executed.

k DESIGNATORS: The address to which the jump is made, Y, is derived as follows:

k = 0 or 4: $Y = \bar{y}$. The address is $\bar{y}$, the low-order 15 bits contained in the instruction word after B-register modification.

k = 1, 3, or 5: $Y = Y_L$. The address is the low-order 15 bits contained in the storage location at address $\bar{y}$.

k = 2 or 6: $Y = Y_U$. The address is the high-order 15 bits contained in the storage location at address $\bar{y}$.

k = 7: $Y = A_L$. The address is the low-order 15 bits contained in the A-register.

j DESIGNATORS: The jump conditions are determined as follows:

j = 0: Execute jump regardless of key settings.

j = 1: Execute jump if JUMP 1 key is set.

j = 2: Execute jump if JUMP 2 key is set.

j = 3: Execute jump if JUMP 3 key is set.

j = 4: Execute jump and stop regardless of key settings.

j = 5: Execute jump. Stop if STOP 5 key is set.

j = 6: Execute jump. Stop if STOP 6 key is set.

j = 7: Execute jump. Stop if STOP 7 key is set.

RETURN JUMP (arithmetic)

CLASS: Read

FUNCTION CODE: 64

MNEMONIC: RJP•

OPERATION: Jump to address $Y + 1$ and $P + 1 \rightarrow Y_L$ if the jump condition is satisfied. If the jump condition is not satisfied, execute *NI*.

DESCRIPTION: This instruction, depending upon the condition of the contents of either the A- or Q-register, transfers program control to another section of the program and stores the address at which the original sequence of instructions may be resumed. If a jump condition is satisfied, a jump is made to address $Y + 1$ and the address of the instruction immediately following the Return Jump instruction, $P + 1$, is stored in the lower-half of the storage location at address Y. If the jump condition is not satisfied, the instruction immediately, following the Return Jump instruction is executed.

k DESIGNATORS: The address of the storage location, Y, the lower-half of which will contain address $P + 1$, is derived as follows:

k = 0 or 4: $Y = \bar{y}$. The address is $\bar{y}$ – the low-order 15 bits contained in the instruction word after B-register modification.

k = 1, 3, or 5: $Y = Y_L$. The address is the low-order 15 bits contained in the storage location at address y.

k = 2 or 6: $Y = Y_U$. The address is the high-order 15 bits contained in the storage location at address $\bar{y}$.

k = 7: $Y = A_L$. The address is the low-order 15 bits contained in the A-register.

j DESIGNATORS: The jump conditions are determined as follows:

j = 0: Do not execute return jump. Set interrupt lockout.

j = 1: Execute return jump. Set interrupt lockout.

j = 2: Execute return jump if *(Q)* is + or + 0.

j = 3: Execute return jump if *(Q)* is − or − 0.

j = 4: Execute return jump if *(A)* is + 0.

j = 5: Execute return jump if *(A)* is not + 0.

j = 6: Execute return jump if *(A)* is + or + 0.

j = 7: Execute return jump if *(A)* is − or − 0.

RETURN JUMP (manual)

CLASS Read

FUNCTION CODE: 65

MNEMONIC: RJP •

OPERATION: Jump to address $Y + 1$ and $P + 1 \rightarrow Y_L$ if the jump condition is satisfied. If the jump condition is not satisfied, execute *NI*.

DESCRIPTION: This instruction, as directed by the key settings on the maintenance panel and/or operator console, transfers program control to another section of the program and stores the address at which the original sequence of instructions may be resumed. If a jump condition is satisfied, a jump is made to address $Y + 1$ and the address of the instruction immediately following the Return Jump instruction, $P + 1$, is stored in the lower-half of the storage location at address Y. In certain cases the key setting will cause a jump and stop operation if the jump condition is satisfied. In these situations, the Computer stops after address $P + 1$ has been stored at address Y with the *P*-register set to execute the instruction stored at address $Y + 1$. In all cases where the jump condition is not satisfied, the instruction immediately following the Return Jump instruction is executed.

k DESIGNATORS: The address of the storage location, Y, the lower-half of which will contain $P + 1$, is derived as follows:

k = 0 or 4: $Y = \bar{y}$. The address is $\bar{y}$ − the low-order 15 bits contained in the instruction word after *B*-register modification.

k = 1, 3, or 5: $Y = Y_L$. The address is the low-order 15 bits contained in the storage location at address $\bar{y}$.

k = 2 or 6: $Y = Y_U$. The address is the high-order 15 bits contained in the storage location at address $\bar{y}$.

k = 7: $Y = A_L$. The address is the low-order 15 bits contained in the *A*-register.

j DESIGNATORS: The jump conditions are determined as follows:

j = 0: Execute return jump regardless of key settings.

j = 1: Execute return jump if JUMP 1 key is set.

j = 2: Execute return jump if JUMP 2 key is set.

j = 3: Execute return jump if JUMP 3 key is set.

j = 4: Execute return jump and stop regardless of key settings.

j = 5: Execute return jump. Stop if STOP 5 key is set.

j = 6: Execute return jump. Stop if STOP 6 key is set.

j = 7: Execute return jump. Stop if STOP 7 key is set.

UTILIZING RETURN JUMP INSTRUCTIONS

Return jump instructions can be used to cause a program to jump to a subroutine, execute the subroutine, and jump back to resume the original sequence of instructions after the subroutine has been completed. This process can be accomplished if the exit jump precedes the first instruction in the subroutine and immediately following the last instruction there is a jump instruction that causes a jump to the exit. The following example illustrates this concept:

Main Program

Address		
L	XXXXXXXXXX	First Instruction
L + 1	XXXXXXXXXX	Second Instruction
L + 2	64000(S)	Reference Subroutine
L + 3	XXXXXXXXXX	Third Instruction

Subroutine

Address		
S	00000 (*P*)	Exit Jump
S + 1	XXXXXXXXXX	First Instruction
.	.	.
.	.	.
.	.	.
S + *n* − 1	XXXXXXXXXX	Last Instruction
S + *n*	61010(S)	Jump to S_L

In this example the operation proceeds as follows:

1. After instructions at addresses L and $L + 1$ in the main program are executed, the *P*-register contains address $L + 2$ (assuming that the instruction at $L + 1$ did not cause a skip or a jump).

2. The 64 instruction at address $L + 2$ causes address $L + 3$ to be stored in the lower-half of the storage location at address S and a jump to be made to address $S + 1$.

3. The instructions at addresses $S + 1$ through $S + n - 1$ are executed.

4. The instruction at address $S + n$ is executed causing a jump to be made to address S_L.

9. INPUT-OUTPUT INSTRUCTIONS

Input-output instructions are used to facilitate the transfer of data between the Computer and the various peripheral subsystems.

INPUT-OUTPUT INSTRUCTION WORD

The format for input-output instruction words is shown in Figure 9-1.

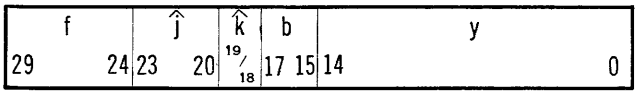

Figure 9-1. Input-Output Instruction Word

f-DESIGNATOR

The function code designator, *f*, is a 6-bit code that specifies the operation to be performed.

y-DESIGNATOR

The operand-designator, *y*, is a 15-bit code that represents either the operand or the operand address.

$\hat{j}$-DESIGNATOR

The channel designator, $\hat{j}$, is a 4-bit code that specifies the input or output channel that the instruction refers to.

$\hat{k}$-DESIGNATOR

The operand-interpretation designator, $\hat{k}$, is a 2-bit code that controls where the operand is procured from or where it is stored, or both.

$\hat{j}\,\hat{k}$ COMBINATIONS

As shown in Figure 9-1, $\hat{j}$ and $\hat{k}$ together occupy the same bit positions as j and k in all other instructions; however, $\hat{j}$ and $\hat{k}$ are 4 bits and 2 bits as opposed to 3 bits and 3 bits for j and k. When input-output instructions are written, the 6-bits that represent the $\hat{j}\,\hat{k}$ combination appear as two octal digits as do the 6-bits representing $j\ k$ in all other instructions. In the case of input-output instructions, these octal digits are considered as a unit that represents a specific $\hat{j}\,\hat{k}$ combination rather than having one digit represent $\hat{j}$ and one represent $\hat{k}$. The octal digits that represent the $\hat{j}\,\hat{k}$ combination are shown at the intersections of the $\hat{j}$ value rows and $\hat{k}$ value columns in Figure 9-2.

For example, assume that an input-output instruction is to be written with the requirement that $\hat{j}=5$ and $\hat{k}=3$. An examination of the intersection of the $\hat{j}=5$ row and the $\hat{k}=3$ column in the diagram will show that 27 is the 2-digit octal combination that meets this requirement.

j-k COMBINATIONS FOR I/O FUNCTIONS

	k = 0	k = 1	k = 2	k = 3
j = 0	00	01	02	03
j = 1	04	05	06	07
j = 2	10	11	12	13
j = 3	14	15	16	17
j = 4	20	21	22	23
j = 5	24	25	26	27
j = 6	30	31	32	33
j = 7	34	35	36	37
$8_D = 10_8$	40	41	42	43
$9_D = 11_8$	44	45	46	47
$10_D = 12_8$	50	51	52	53
$11_D = 13_8$	54	55	56	57
$12_D = 14_8$	60	61	62	63
$13_D = 15_8$	64	65	66	67

Figure 9-2. $\hat{j}$ $\hat{k}$ Combination For Input-Output Instructions

b-DESIGNATOR

The operand address modification designator, *b*, a 3-bit code, specifies the *B*-register containing a quantity that is added to the operand address.

INPUT-OUTPUT BUFFERS

An input buffer is a block of consecutive storage locations into which a peripheral subsystem, connected to an input channel, places data. Conversely, an output buffer is a block of consecutive storage locations from which a peripheral subsystem, connected to an output channel, receives data. The assignment of the buffers is made by the buffer instructions (73, 74, 75, and 76). These instructions activate a buffer and place a control word in the appropriate buffer-control register. The control word contains two addresses that define the size and location of the buffer. Figure 9-3 shows the format of the control word.

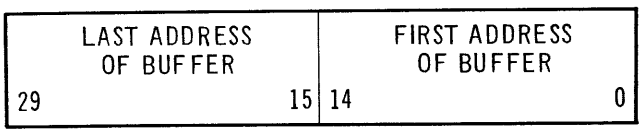

Figure 9-3. Control Word

As shown in the diagram, the high-order 15 bits of the control word contain the address of the last word in the buffer and the low-order 15 bits contain the address of the first word.

There are 14 input channels numbered 0 through 15 (octal) and 14 output channels numbered 0 through 15 (octal). Channels 0 and 1 are reserved for computer-to-computer communication. For each of these channels a fixed storage location is designated as a buffer-control register. The input buffer-control registers are located at octal addresses 00100 through 00115 and the output buffer-control registers at octal addresses 00120 through 00135. The buffer-control register for a particular channel is determined by the $\hat{j}$ designator in the buffer instruction. Since $0 \leq \hat{j} \leq 15$, the input buffer-control register for channel $\hat{j}$ is at address $00100 + \hat{j}$ and the output buffer-control register is at address $00120 + \hat{j}$.

At the time a buffer is activated, the lower-half of the appropriate buffer-control register contains the first address of the buffer and the upper-half contains the last address. For example, if data is transferred on input channel 12 to a five-word input buffer located at addresses 01000, 01001, 01002, 01003, and 01004, the input buffer-control register, for this channel, located at address 00112, contains 0100401000.

As the buffer operates, 30-bit data words are transferred to or from the consecutive addresses in the buffer at a rate determined by the peripheral subsystem attached to the input or output channel. The first transfer is made to or from the storage location whose address is contained in the lower-half of the buffer-control register. When the first transfer is completed, the lower-half of the buffer-control register is incremented by 1 to contain the address to which the next data word will be transferred to or from. These operations continue to transfer and increment until the buffer is filled or emptied. At this point the buffer operation is terminated and the lower-half of the buffer-control register contains the address of the first storage location beyond the buffer; that is, the last address of the buffer + 1.

For example, data is being transferred from a peripheral subsystem on input channel 11 to a 5-word input buffer located at addresses 01000, 01001, 01002, 01003, and 01004. In this case, the buffer operation would be as follows:

1. The initial contents of the input buffer-control register at address 00112 will be 0100401000.

2. Thirty-bit data words will be transferred to addresses 01000 through 01004 and the buffer-control register will be incremented following each transfer.

3. When the buffer mode is terminated, the input buffer-control register will contain 0100401005.

ACTIVATE INPUT BUFFER WITHOUT MONITOR

CLASS: Read

FUNCTION CODE: 73

MNEMONIC: IN · Cn

OPERATION: Activate input buffer on input channel $\hat{j}$ without monitor.

DESCRIPTION: This instruction activates an input buffer on input channel $\hat{j}$ and sets up the appropriate input buffer-control register by placing a control word that defines the size and location of the buffer, at address 00100 + $\hat{j}$.

$\hat{k}$ DESIGNATORS: The input buffer control register is set up as follows:

$\hat{k}$ = 0: $\overline{y}$, the low-order 15 bits contained in the instruction word after *B*-register modification, is placed in the lower-half of the control register thus establishing the first address of the buffer. The upper-half of the control register is undisturbed; consequently, the high-order 15 bits contained there represent the last address of the buffer.

$\hat{k}$ = 1: Y_L, the low-order 15 bits contained in the storage location at address $\overline{y}$, is placed in the lower-half of the control register thus establishing the first address of the buffer. The upper-half of the control register is undisturbed; consequently, the high-order 15 bits contained there represent the last address of the buffer.

$\hat{k}$ = 2: Not used.

$\hat{k}$ = 3: *Y*, the 30 bits contained in the storage location at address $\overline{y}$, is placed in the control register thus establishing the first and last address of the buffer.

$\hat{j}$-DESIGNATORS: All values for $\hat{j}$ are permissible with this instruction.

$\hat{j}\,\hat{k}$ COMBINATIONS: Refer to Figure 9-2.

ACTIVATE OUTPUT BUFFER WITHOUT MONITOR

CLASS: Read

MNEMONIC: OUT · Cn

OPERATION: Activate output buffer on output channel $\hat{j}$ without monitor.

DESCRIPTION: This instruction activates an output buffer on output channel $\hat{j}$ and sets up the appropriate output buffer-control register by placing a control word that defines the size and location of the buffer at address 00120 + $\hat{j}$.

$\hat{k}$ DESIGNATORS: The output-buffer-control register is set up as follows:

$\hat{k}$ = 0: $\overline{y}$, the low-order 15 bits contained in the instruction word after *B*-register modification, is placed in the lower-half of the control register thus establishing the first address of the buffer. The upper-half of the control register is undisturbed; consequently, the high-order 15 bits contained there represent the last address of the buffer.

$\hat{k}$ = 1: Y_L, the low-order 15 bits contained in the storage location at address $\overline{y}$, is placed in the lower-half of the control register thus establishing the first address of the buffer. The upper-half of the control register is undisturbed; consequently, the high-order 15 bits contained there represent the last address of the buffer.

$\hat{k}$ = 2: Not used.

$\hat{k}$ = 3: *Y*, the 30 bits contained in the storage location at address $\overline{y}$, is placed in the control register thus establishing the first and last address of the buffer.

$\hat{j}$ DESIGNATORS: All values for $\hat{j}$ are permissible with this instruction.

$\hat{j}\,\hat{k}$ COMBINATIONS: Refer to Figure 9-2.

ACTIVATE INPUT BUFFER WITH MONITOR

CLASS: Read

FUNCTION CODE: 75

MNEMONIC: IN · Cn · MONITOR

OPERATION: Activate input buffer on input channel $\hat{j}$ with monitor.

DESCRIPTION: This instruction activates an input buffer on input channel $\hat{j}$ and sets up the appropriate input buffer-control register by placing a control word that defines the size and location of the buffer at address 00100 + $\hat{j}$. When the buffer is filled, an internal interrupt will occur on input channel $\hat{j}$.

$\hat{k}$ DESIGNATORS: The input buffer control register is set up as follows:

$\hat{k}$ = 0: $\overline{y}$, the low-order 15 contained bits in the instruction word after *B*-register modification, is placed in the lower-half of the control register thus establishing the first address of the buffer. The upper-half of the control register is undisturbed; consequently, the high-order 15 bits contained there represent the last address of the buffer.

$\hat{k}$ = 1: Y_L, the low-order 15 bits contained in the storage location at address $\overline{y}$, is placed in the lower-half of the control register thus establishing the first address of the buffer. The upper-half of the control register is undisturbed; consequently, the high-order 15 bits contained there represent the last address of the buffer.

$\hat{k}$ = 2: Not used.

$\hat{k}$ = 3: *Y*, the 30 bits contained in the storage location at address $\overline{y}$, is placed in the control register thus establishing the first and last address of the buffer.

$\hat{j}$ DESIGNATORS: All values for $\hat{j}$ are permissible with this instruction.

$\hat{j}$ $\hat{k}$ COMBINATIONS: Refer to Figure 9-2.

ACTIVATE OUTPUT BUFFER WITH MONITOR

CLASS: Read

FUNCTION CODE: 76

MNEMONIC: OUT · Cn · MONITOR

OPERATION: Activate output buffer on output channel $\hat{j}$ with monitor.

DESCRIPTION: This instruction activates an output buffer an output channel $\hat{j}$ and sets up the appropriate output buffer-control register by placing a control word that defines the size and location of the buffer at address 00120 + $\hat{j}$. When the buffer is emptied, an internal interrupt will occur on output channel $\hat{j}$.

$\hat{k}$ DESIGNATORS: The output buffer-control register is set up as follows:

$\hat{k}$ = 0: $\overline{y}$, the low-order 15 bits contained in the instruction word after *B*-register modification, is placed in the lower-half of the control register thus establishing the first address of the buffer. The upper-half of the control register is undisturbed; consequently, the high-order 15 bits contained there represent the last address of the buffer.

$\hat{k}$ = 1: Y_L, the low-order 15 bits contained in the storage location at address $\overline{y}$, is placed in the lower-half of the control register thus establishing the first address of the buffer. The upper-half of the control register is undisturbed; consequently, the high-order 15 bits contained there represent the last address of the buffer.

$\hat{k}$ = 2: Not used.

$\hat{k}$ = 3: *Y*, the 30 bits contained in the storage location at address $\overline{y}$, is placed in the control register thus establishing the first and last address of the buffer.

$\hat{j}$ DESIGNATORS: All values for $\hat{j}$ are permissible with this instruction.

$\hat{j}$ $\hat{k}$ COMBINATIONS: Refer to Figure 9-2.

JUMP ON ACTIVE INPUT BUFFER

CLASS: Read

FUNCTION CODE: 62

MNEMONIC: JP · Cn · ACTIVEIN

OPERATION: Jump to address *Y* if the input buffer on input channel $\hat{j}$ is active. If the input buffer is not active, execute *NI*.

DESCRIPTION: This instruction transfers program control to another section of the program if the input buffer on input channel $\hat{j}$ is active. If the input buffer is active, a jump will be made to address *Y*. If the input buffer is not active, the instruction immediately following the Jump instruction is executed.

$\hat{k}$ DESIGNATORS: The address to which the jump is made, *Y*, is derived as follows:

$\hat{k} = 0$: $Y = \bar{y}$. The address is $\bar{y}$ – the low-order 15 bits contained in the instruction word after *B*-register modification.

$\hat{k} = 1$ or 3: $Y = Y_L$. The address is the low-order 15 bits contained in the storage location at address $\bar{y}$.

$\hat{k} = 2$: $Y = Y_U$. The address is the high-order 15 bits contained in the storage location at address $\bar{y}$.

$\hat{j}$ DESIGNATORS: All values for $\hat{j}$ are permissible with this instruction.

$\hat{j}$ $\hat{k}$ COMBINATIONS: Refer to Figure 9-2.

JUMP ON ACTIVE OUTPUT BUFFER

CLASS: Read

FUNCTION CODE: 63

MNEMONIC: JP • Cn • ACTIVEOUT

OPERATION: Jump to address *Y* if the output on output channel $\hat{j}$ is active. If the output buffer is not active, execute *NI*.

DESCRIPTION: This instruction transfers program control to another section of the program if the output buffer on output channel $\hat{j}$ is active. If the output buffer is active, a jump will be made to address *Y*. If the output buffer is not active, the instruction immediately following the Jump instruction is executed.

$\hat{k}$ DESIGNATORS: The address to which the jump is made, *Y*, is derived as follows:

$\hat{k} = 0$: $Y = \bar{y}$. The address is $\bar{y}$ – the low-order 15 bits contained in the instruction word after *B*-register modification.

$\hat{k} = 1$ or 3: $Y = Y_L$. The address is the low-order 15 bits contained in the storage location at address $\bar{y}$.

$\hat{k} = 2$: $Y = Y_U$. The address is the high-order 15 bits contained in the storage location at address $\bar{y}$.

$\hat{j}$ DESIGNATORS: All values for $\hat{j}$ are permissible with this instruction.

j k COMBINATIONS: Refer to Figure 9-2.

TERMINATE INPUT BUFFER

CLASS: Read

FUNCTION CODE: 66

MNEMONIC: TERM • Cn • INPUT

OPERATION: Terminate input buffer on input channel $\hat{j}$.

DESCRIPTION: This instruction terminates the input buffer on input channel $\hat{j}$. The transfer currently in process is completed and no further transfers are made to the buffer. If the buffer was activated with monitor, the internal interrupt that would normally follow after the buffer was filled will not occur. The last buffer address to which a transfer was made can be determined by examining the lower-half of the input buffer-control register at address 00100 + $\hat{j}$ and subtracting 1 from the address that it contains. The low-order 20 bits of the instruction word, the $\hat{k}$, *b*, and *y* designators, do not affect the execution of this instruction.

$\hat{k}$ DESIGNATORS: All values for $\hat{k}$ are permissible with this instruction.

$\hat{j}$ DESIGNATORS: All values for $\hat{j}$ are permissible with this instruction.

$\hat{j}$ $\hat{k}$ COMBINATIONS: Refer to Figure 9-2.

TERMINATE OUTPUT BUFFER

CLASS: Read

FUNCTION CODE: 67

MNEMONIC: TERM • Cn • OUTPUT

OPERATION: Terminate output buffer on output channel $\hat{j}$.

DESCRIPTION: This instruction terminates the output buffer on output channel $\hat{j}$. The transfer currently in process is completed and no further transfers are made from the buffer. If the buffer was activated with monitor, the internal interrupt that would normally follow after the buffer was emptied will not occur. The last buffer address that a transfer was made from can be determined by examining the lower-half of the output-buffer control register at address 00120 + $\hat{j}$ and subtracting 1 from the address that it contains. The low-order 20 bits of the instruction word, the $\hat{k}$, *b*, and *y*-designators, do not affect the execution of this instruction.

$\hat{k}$ DESIGNATORS: All values for $\hat{k}$ are permissible with this instruction.

$\hat{j}$ DESIGNATORS: All values for $\hat{j}$ are permissible with this instruction.

$\hat{j}$ $\hat{k}$ COMBINATIONS: Refer to Figure 9-2.

FUNCTION WORDS

Function Words are commands that are sent to a peripheral subsystem that is connected to a particular channel. These Function Words cause the initiation or termination of some action by the peripheral subsystem. The formats of the Function Words for the various peripheral units are shown below:

Drum

The format of the Function Word for the drum is shown in Figure 9-4.

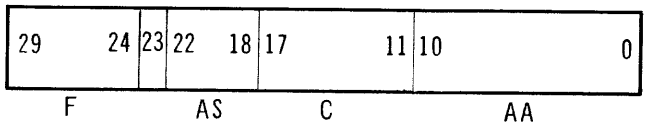

Figure 9-4. Drum Function Word

As shown in the diagram, bit positions 0 through 10 (the *AA* portion) specify the angular address, bit positions 11 through 17 (the *C* portion) specify the channel, bit positions 18 through 22 (the *AS* portion) specify the angular section, and bit positions 24 through 29 (the *F* portion) specify the function code. Bit 23 is not utilized; consequently, a 0 or a 1 may be placed in this bit position.

Magnetic Tape

The format of the Function Word for magnetic tape is shown in Figure 9-5.

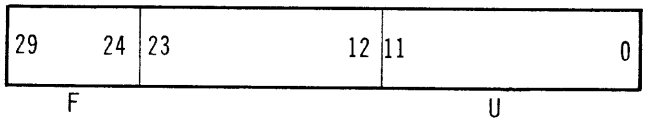

Figure 9-5. Magnetic Tape Function Word

As shown in the diagram, bit positions 0 through 11 (the *U* portion) specify the unit, and bit positions 24 through 29 (the *F* portion) specify the function code. Bit positions 12 through 23 are not utilized; consequently, any combination of bits may appear in these bit positions.

Card

The format of the Function Word for punched cards is shown in Figure 9-6.

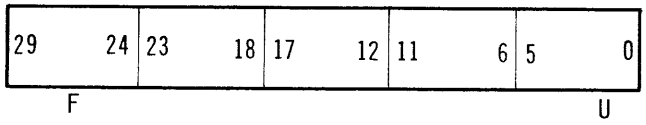

Figure 9-6. Punched Card Function Word

As shown in the diagram, bit positions 24 through 29 (the *F* portion) specify the function code. Bit positions 0 through 23 are not utilized; consequently, any combination of bits may appear in these bit positions.

High-Speed Printer

The format of the Function Word for the High-Speed Printer is shown in Figure 9-7.

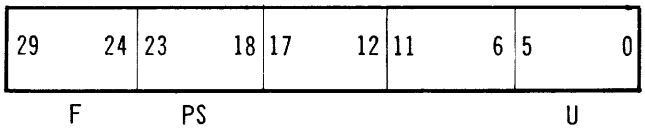

Figure 9-7. High-Speed Printer Function Word

As shown in the diagram, bit positions 0 through 5 (the *U* portion) specify the unit, bit positions 18 through 23 (the *PS* portion) specify the paper spacing (0 through 63 lines), and bit positions 24 through 29 (the *F* portion) specify the function code. Bit positions 6 through 17 are not utilized; consequently, any combination of bits may appear in these bit positions.

Paper Tape

The format of the Function Word for paper tape is shown in Figure 9-8.

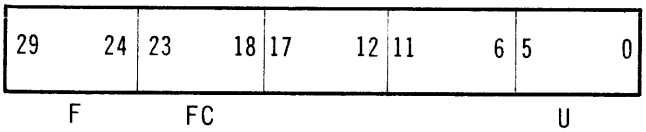

Figure 9-8. Paper Tape Function Word

As shown in the diagram, bit positions 0 through 5 (the *U* portion) specify the unit, bit positions 18 through 23 (the *FC* portion) specify the format control (6 or 8 level), and bit positions 24 through 29 (the *F* portion) specify the function code. Bit positions 6 through 17 are not utilized; consequently, any combination of bits may appear in these bit positions.

Function Codes

The function codes (the *F* portions in the Function Words) determine the action that a peripheral unit will take. Figure 9-9 provides a listing of these codes and the actions they perform.

OCTAL CODE	DESCRIPTION OF FUNCTION
01	a. Write one block on magnetic tape at 12.5 kc.
	b. Punch Paper Tape.
02	a. Write one block on magnetic tape at 25 kc.
	b. Print on line printer at 6 lines per inch.
	c. Write continuously on Magnetic Drum.
	d. Punch on Card Punch Unit
	e. Punch on Paper Tape Punch with parity.
03	a. Print on line printer at 8 l.p.i. (Model 151 only)
11	Same as 01 function word except at end of function, an EXT. INT. will be sent to computer.
12	Same as 02 function word except at end of function, an EXT. INT. will be sent to computer.
20	a. Rewind Magnetic Tape.
21	a. Rewind Magnetic Tape with interlock.
23	a. Terminate present function.
30,31,	Same as 20,21, except: EXT.INT. will be sent to computer at end of function.
40	Automatic Bootstraps a. Magnetic Tape
	b. Drum
	c. Paper Tape
41	a. Read Forward Magnetic Tape one block, low gain.
	b. Read forward Paper Tape
42	a. Read Forward Magnetic Tape, normal gain.
	b. Read continuously, Magnetic Drum.
	c. Read Paper Tape – check parity
	d. Read from Typewriter
	c. Read from Card Reader.
43	a. Read Forward Magnetic Tape, high gain.
45	a. Search Read Mag. Tape. After "find" read block, low gain.
	b. Search on Magnetic Drum and transfer address of "find" word of "find" word.
46	a. Search Read Magnetic Tape. After "find", read block, normal gain.
	b. Search Read Magnetic Drum.
47	a. Search Read Magnetic Tape. After "find", read block, high gain.
50,51,52, 53,55,56, 57	Same as 40 series except EXT. INT. sent to computer at end of function. On Magnetic Drum means present function will be ended by end-of-block word.
61	a. Read Magnetic Tape backward at low gain.
	b. Read Paper Tape backward.
62	a. Read Magnetic Tape backward, normal gain.
	b. Read Paper Tape backward checking parity.
63	a. Read Magnetic Tape backward, high gain.
65	a. Search read on Magnetic Tape backward, low gain.
66	a. Search read on Magnetic Tape backward, normal gain.
67	a. Search read on Magnetic Tape backward, high gain.
71,72,73	Same as corresponding 60 series except that at the completion of the function, at EXT. INT. is sent to the computer.

Figure 9-9. Input-Output Function Codes

INTERNAL INTERRUPT

An internal interrupt occurs when a buffer that was activated with monitor is filled or emptied. At this point, the program is interrupted and instead of executing the next instruction in the program, an instruction contained in the appropriate internal interrupt entrance register is executed. The input internal interrupt entrance registers are located at addresses 00040 through 00055, and the output internal interrupt entrance registers are located at addresses 00060 through 00075. Since $0 \leq \hat{j} \leq 15$, the input internal interrupt entrance register for channel $\hat{j}$ is at address 00040 + $\hat{j}$ and the output internal interrupt entrance register is at address 00060 + $\hat{j}$. These registers should contain instructions that will transfer control to a subroutine that performs the action required by the internal interrupt and when this action has been completed causes the program to be resumed at the point where it was interrupted.

Since the program address register, *P*, contains the address of the next instruction at the time that the interrupt occurs, the process described above can be accomplished if the programmer has previously stored a 65 instruction with *j*, *k*, and b = 0 in the appropriate internal interrupt entrance register.

When an internal interrupt occurs, all other interrupts other than error interrupts will be locked-out until a 60 instruction with *j* = 0 or 1 has been executed.

EXTERNAL INTERRUPT

An external interrupt is a program interrupt initiated by an external device. A peripheral subsystem connected to any one of the input channels (channels 0 through 15) may interrupt the current program by sending an external interrupt of the Computer. When this occurs, the current program is interrupted and instead of executing the next instruction in the current program an instruction contained in the appropriate external interrupt entrance register is executed. The external interrupt registers are located at addresses 00020 through 00035. Since $0 \leq \hat{j} \leq 15$ the external interrupt register for channel $\hat{j}$ is located at address 00020 + $\hat{j}$.

When an external interrupt occurs, the procedure outlined for internal interrupts applies with the exception that the subroutine to which control is transferred should contain a 17 instruction.

External interrupts like internal interrupts cause all (except error) interrupts to be locked-out until a 60 instruction with $j = 0$ or 1 is executed.

External Function

CLASS: Read

FUNCTION CODE: 13

MNEMONIC: If $\hat{j} \neq 0$ or 1, EX • FCT • Cn
If $\hat{j} = 0$ or 1, TEST • CO; C1

OPERATION: If $\hat{j} \neq 0$ or 1, Y⟶*Cj*.
If $\hat{j} = 0$ or 1, skip *NI* if input buffer on channel $\hat{j}$ is active; if inactive, execute *NI*.

DESCRIPTION: If $\hat{j} \neq 0$ or 1, this instruction transmits a 30-bit Function Word via the selected output channel to a peripheral subsystem. The Function Word will cause the initiation or termination of some action by the peripheral unit that is connected to the particular output channel. If $\hat{j} = 0$ or 1, this instruction will determine if the input buffer for the selected computer-to-computer channel is active. If the buffer is active, the instruction immediately following the External Function instruction is skipped; if inactive this instruction is executed. When $\hat{j} = 0$ or 1 bit positions 0 through 18, the *k*- and *y*-designators, do not affect the execution of this instruction; consequently, any combination of bits may appear in these bit positions.

$\hat{k}$ DESIGNATORS: In both cases only $k = 3$ is permitted. If $\hat{j} \neq 0$ or 1, the 30-bit Function Word contained in the storage location at address $\bar{y}$ is transmitted via the selected channel to a peripheral unit. If $\hat{j} = 0$ or 1, $\hat{k} = 3$ is required for timing purposes.

$\hat{j}$ DESIGNATORS: All values for $\hat{j}$ are permissible with this instruction.

$\hat{j}$ $\hat{k}$ COMBINATIONS: Refer to Figure 9-2.

Store C Channel

CLASS: Store

FUNCTION CODE: 17

MNEMONIC: STR • Cn

OPERATION: *(C)j*⟶*Y*

DESCRIPTION: This instruction stores a 30-bit external interrupt code for input channel $\hat{j}$ in a storage location. It should only be used following an external interrupt.

$\hat{k}$ DESIGNATORS: Only $\hat{k} = 3$ is permitted with this instruction. The external interrupt code for channel $\hat{j}$ will be stored in the storage location at address $\bar{y}$.

$\hat{j}$ DESIGNATORS: All values for $\hat{j}$ except 0 or 1 may be used with this instruction.

$\hat{j}$ $\hat{k}$ COMBINATIONS: Refer to Figure 9-2.

INPUT-OUTPUT PRIORITY STRUCTURE

The cycle that transfers one 30-bit word from or to the Computer via a channel is called a buffer action. Only one buffer action can be performed at a time; however, several operating channels may simultaneously request buffer action. Therefore, the input-output control logic assigns different priorities to each of the buffer actions. The buffer actions in order or priority from highest to lowest are listed below:

1. *Incremental Interrupt Clock Update:* This action occurs approximately once each millisecond and requires one memory cycle during which time no other buffer action may take place.

2. *External Interrupt:* The Computer program will process this interrupt ignoring all other interrupts until a 60 instruction with $j = 0$ or 1 is executed. Other data transfer requests on the same channel cannot be made until a 17 instruction is executed which allows the interrupting device to drop its interrupt signal.

3. *Output Internal Interrupt:* All other interrupts are ignored until a 60 instruction with $j = 0$ or 1 is executed. Other data transfer requests cannot be made on this channel until a buffer instruction that activates the transfer logic is executed.

4. *Input Internal Interrupt:* All other interrupts are ignored until a 60 instruction with $j = 0$ or 1 is executed. Other data transfer requests cannot be made on this channel until a buffer instruction that activates the transfer logic is executed.

5. *Output Buffer Word Transfer:* This action, when started, allows the transfer of a single word out of the Computer memory during which time no other buffer actions may take place.

6. *Input Buffer-Word Transfer:* This action, when started, allows the transfer of a single word into the Computer memory during which time no other buffer action may take place.

7. *Computer-To-Computer-Output Transfer:* This action allows the transfer of one word out of one of the two computer-to-computer channels before re-examining the priority status of other input-output operations.

8. *Programmed Input-Output Instructions:* These instructions, when initiated, lock out other input-output operations for the duration of the instruction.

Each of the above operations has an additional priority associated with it; that is, the operation with the highest priority that is present on the highest numbered channel will be performed first.

Remington Rand Univac

DIVISION OF SPERRY RAND CORPORATION

PRINTED IN U.S.A. UT2452