NAVAL TACTICAL DATA SYSTEM (NTDS)

technical note no. 240

REPERTOIRE of INSTRUCTIONS for the AN/USQ-20 UNIT COMPUTER

Remington Rand Univac
DIVISION OF SPERRY RAND CORPORATION
UNIVAC PARK, ST. PAUL 16, MINNESOTA

NAVAL TACTICAL DATA SYSTEM
TECHNICAL NOTE
NO. 240

REPERTOIRE of INSTRUCTIONS
for the
AN/USQ-20 UNIT COMPUTER

by

Walter G. Haberstroh

PX 1343-36

Remington Rand Univac®
DIVISION OF SPERRY RAND CORPORATION
UNIVAC PARK, ST. PAUL 16, MINNESOTA

NAVY DEPARTMENT | BUREAU OF SHIPS | ELECTRONICS DIVISIONS
CONTRACT: NObsr 72769 | NTDS NO. U-6090 | 1 AUGUST 1960

CONTENTS

TECHNICAL NOTE NO. 240

REPERTOIRE OF INSTRUCTIONS FOR THE AN/USQ-20 UNIT COMPUTER

1. INTRODUCTION

This technical note presents the instruction repertoire for the AN/USQ-20 NTDS Unit Computer. Details presented are limited to the needs of the NTDS programmer and list only symbols, registers, terms, and instruction characteristics pertinent to programming the computer.

Major programming differences between the AN/USQ-20 and its forerunner, the AN/USQ-17, lie in the area of input/output characteristics. The AN/USQ-20 Unit Computer features simplified instructions pertaining to both input and output on 14D channels, 12D of which have external function capabilities. In addition, the computer provides a more powerful *Repeat* instruction, two instructions with parity check, buffer monitoring, and other modifications.

Those familiar with the AN/USQ-17 instruction repertoire should make special note of the following AN/USQ-20 instructions: 13, 17, 40, 44, 60, 62, 63, 66, 70, 73, 74, 75, and 76. Major revisions have been made to these instructions specifically. In addition, the reader must also be aware of fault procedures since function codes 00 and 77 are *fault conditions* which, if executed, will cause a *fault interrupt.*

2. GENERAL INFORMATION

The Naval Tactical Data System Unit Computer (NTDSUC) is a self-modifying, one-address computer. Although this means that one reference or address is provided for the execution of an instruction, this reference can be modified automatically during a programmed sequence. The references are modified by using the B (index) registers one through seven, which contain any previously stored constants. To modify the address, the content of a selected B-register is added to the Operand Designator, **y**.

A programmed address is coded using octal notation with each octal digit denoting three binary digits. The instructions are read sequentially from Magnetic Core Storage except after Jump or Skip instructions.

A. *SYMBOL CONVENTIONS* - The following symbols are used throughout the descriptive material on instructions:

α	=	a register (A, Q, B^n), a memory location Y, or a constant.
(α)	=	content of α.
$(\alpha)_i$	=	initial content of α.
$(\alpha)_f$	=	final content of α.
α_n	=	the n^{th} bit of α.
$(\alpha)_n$	=	the n^{th} bit of the content of α.
f	=	Function Code Designator $(i_{29}, \ldots, i_{24})$*.
j	=	Branch Condition Designator $(i_{23}, \ldots, i_{21})$*.
$\hat{\mathbf{j}}$	=	Input/Output Channel Designator $(i_{23}, \ldots, i_{20})$*.
k	=	Operand Interpretation Designator $(i_{20}, \ldots, i_{18})$*.
$\hat{\mathbf{k}}$	=	Operand Interpretation Designator $(i_{19}, \ldots, i_{18})$*.
b	=	Index Designator (i_{17}, i_{16}, i_{15})*.
y	=	Operand Designator $(i_{14}, \ldots, i_0)$*.
Y	=	the Operand (regardless of source).
Y	=	**y** + (B^b).

1) The operand or address of the operand for the *Read* portion of an instruction or

2) The destination address for the *Store* portion of an instruction.

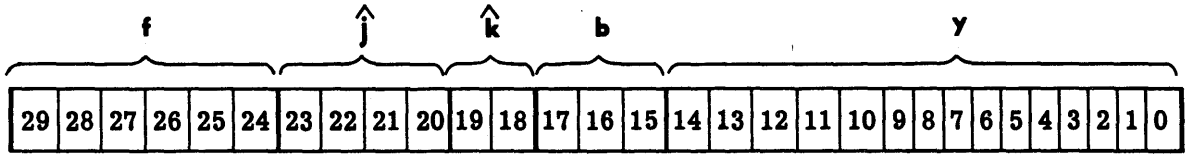
(Y)	=	content of memory address Y.
L(Y)(Q)	=	bit-by-bit multiplication, logical multiply of $\mathbf{Y}_n$, and $(Q)_n$.
A	=	A-register or accumulator (30-bit arithmetic register).
B	=	seven B-registers (15 bits each). B-registers are address-modifying registers generally used for indexing loops in a program; in addition, B^7 serves as a *repeat* counter. (The address modification does not alter the instructions as stored in memory.) A **b** or **j** designator specifies the B-register used.
Q	=	Q-register (30-bit arithmetic register).
U	=	U-register (30 bits). The U-register holds the instruction word during execution of an operation. If address modification is required before execution, the appropriate B-register content is added to the lower-order 15 bits of the U-register before execution.

* i_n is the n^{th} bit position in an instruction.

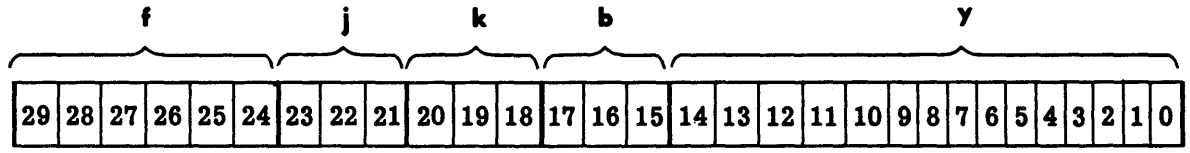
P = P-register (15 bits). The P-register is the Program Address Register. This register holds the address of the current instruction throughout the program except for Jump instructions where the P-register is cleared and the new program address is entered.

C = the 14D input/output channels (30 lines each). Channels consist of transmission lines, therefore they *cannot* be considered registers. The designator $\hat{j}$ specifies (in octal) the channel used.

Figure 1 illustrates bit configuration of instruction designators in two forms. Form I pertains to input/output instructions; Form II pertains to all other instructions.



Form I - Input/Output Instructions



Form II - All Other Instructions

Note: $\hat{j} = C^n$ input/output channel

Figure 1. Bit Allocation of Instruction Designators

Table 1 is a list of the computer's entire repertoire of instructions; each instruction is listed by its function code number and name. Two cards contained in envelopes in the back of this technical note also show the repertoire. In addition, the instructions on these cards are lettered in a form which indicates coding used by the CS-1 Compiling System.*

* Additional references:
1) NTDS Technical Note No. 202 Compiling System CS-1.
2) Compiling System CS-1 - Programmer's Reference Manual PX 1349.
3) Phase III Basic Input Language PX 1478.

TABLE 1. INSTRUCTION REPERTOIRE - AN/USQ-20 UNIT COMPUTER

CODE	FUNCTION NAME	CODE	FUNCTION NAME
00	(Fault Interrupt)	40	ENTER LOGICAL PRODUCT
01	RIGHT SHIFT Q	41	ADD LOGICAL PRODUCT
02	RIGHT SHIFT A	42	SUBTRACT LOGICAL PRODUCT
03	RIGHT SHIFT AQ	43	COMPARE MASKED
04	COMPARE	44	REPLACE LOGICAL PRODUCT
05	LEFT SHIFT Q	45	REPLACE A + LOGICAL PRODUCT
06	LEFT SHIFT A	46	REPLACE A - LOGICAL PRODUCT
07	LEFT SHIFT AQ	47	STORE LOGICAL PRODUCT
10	ENTER Q	50	SELECTIVE SET
11	ENTER A	51	SELECTIVE COMPLEMENT
12	ENTER B^n	52	SELECTIVE CLEAR
13	EXTERNAL FUNCTION ON C^n	53	SELECTIVE SUBSTITUTE
14	STORE Q	54	REPLACE SELECTIVE SET
15	STORE A	55	REPLACE SELECTIVE COMPLEMENT
16	STORE B^n	56	REPLACE SELECTIVE CLEAR
17	STORE C^n	57	REPLACE SELECTIVE SUBSTITUTE
20	ADD A	60	JUMP (Arithmetic)
21	SUBTRACT A	61	JUMP (Manual)
22	MULTIPLY	62	JUMP ON C^n ACTIVE INPUT BUFFER
23	DIVIDE	63	JUMP ON C^n ACTIVE OUTPUT BUFFER
24	REPLACE A + Y	64	RETURN JUMP (Arithmetic)
25	REPLACE A - Y	65	RETURN JUMP (Manual)
26	ADD Q	66	TERMINATE C^n INPUT BUFFER
27	SUBTRACT Q	67	TERMINATE C^n OUTPUT BUFFER
30	ENTER Y + Q	70	REPEAT
31	ENTER Y - Q	71	B SKIP ON B^n
32	STORE A + Q	72	B JUMP ON B^n
33	STORE A - Q	73	INPUT BUFFER ON C^n (without Monitor mode)
34	REPLACE Y + Q	74	OUTPUT BUFFER ON C^n (without Monitor mode)
35	REPLACE Y - Q	75	INPUT BUFFER ON C^n (with Monitor mode)
36	REPLACE Y + 1	76	OUTPUT BUFFER ON C^n (with Monitor mode)
37	REPLACE Y - 1	77	(Fault Interrupt)

B. *FUNCTION CODE DESIGNATOR* - f

The f designator (6 bits) appears in bit-positions 29 through 24 of the U-register, or an instruction, designating the function to be performed by that instruction. All values of f other than 00 and 77 are defined in the instruction list. The two codes 00 and 77 are *fault conditions* which, if executed, will cause a *fault interrupt*. This results in a jump to address 00014, the Fault Entrance Register or address 00014 of *wired* memory depending on the Automatic Recovery Switch setting (see page 12).

C. *BRANCH CONDITION DESIGNATOR* - j

The j designator (3 bits) appears in bit-positions 23, 22, and 21 of the U-register, or an instruction; it is used in a majority of the instructions (see Figure 1, Form II). There are three primary categories of use: 1) for Jump and Skip determination, 2) for B-register specification, and 3) for repeat status interpretation. Appropriate interpretations of the j designator are listed either below or under the descriptions of the individual instructions.

For those instructions in which the j designator has no special interpretation, it specifies the condition under which the next sequential instruction in the program will be skipped. This provides for branching from a sequence without executing a Jump instruction, as would normally occur if a Skip condition were not satisfied.

Skip of the next sequential instruction is determined by the following rules in all instructions except 04, 12, 13, 16, 17, 26, 27, 60 through 67, and 70 through 76.

j = 0: Do not skip the next instruction.
j = 1: Skip the next instruction.
j = 2: Skip the next instruction if (Q) is positive.
j = 3: Skip the next instruction if (Q) is negative.
j = 4: Skip the next instruction if (A) is zero.*
j = 5: Skip the next instruction if (A) is nonzero.
j = 6: Skip the next instruction if (A) is positive.
j = 7: Skip the next instruction if (A) is negative.

When the branch (Skip or Jump) condition involves the sign of the quantity in A or Q, the evaluation examines the sign bit of these quantities; hence, a positive zero (all *zeros*) is considered a positive quantity, and a negative zero (all *ones*) is considered a negative quantity.

---

* Positive zero

D. *INPUT/OUTPUT CHANNEL DESIGNATOR* - $\hat{j}$

The $\hat{j}$ designator (4 bits) appears in bit-positions 23, 22, 21, and 20 of the U-register, or an input/output instruction, specifying the C-channel for the instruction (see Figure 1, Form I). Bit 23 assumes a value of eight, bit 22 a value of four, bit 21 a value of two, and bit 20 a value of one; thus the $\hat{j}$ designator provides accessibility to the 14 (decimal) input/output channels numbered $0\text{-}15_8$.

Instructions 13, 17, 62, 63, 66, 67, 73, 74, 75, and 76 use the $\hat{j}$ designator configuration.

E. *OPERAND INTERPRETATION DESIGNATOR* - k or $\hat{k}$

The k designator (3 bits) [or $\hat{k}$ designator (2 bits)] appears in bit-positions 20, 19, and 18 of the U-register, or an instruction; a $\hat{k}$ designator appears only in bit positions 19 and 18, since bit 20 is a portion of the $\hat{j}$ designator. (See Figure 1, Forms I and II.) Instructions 13, 17, 62, and 73 through 76 use the $\hat{k}$ designator configuration since they perform input/output activities and require a $\hat{j}$ designator for channel specification.

The k and $\hat{k}$ designators control operand interpretation. Those instructions which *read* an operand but do not replace it after the arithmetic is performed are designated *Read* instructions. Those instructions which do not *read* an operand but *store* one are designated *Store* instructions. Instructions which both *read* and *store* operands are classified as *Replace* instructions.

The various values of k or $\hat{k}$ affect the operand in the following list except where otherwise noted under individual instruction descriptions.

1) *Read* instructions (01 through 13, 20 through 23, 26, 27, 30, 31, 40 through 43, 50 through 53, and 60 through 76):

   k or $\hat{k}$ = 0: Y_u = 0's; Y_L = Y.

   k or $\hat{k}$ = 1: Y_u = 0's; $Y_L = (Y)_L$.

   k or $\hat{k}$ = 2: Y_u = 0's; $Y_L = (Y)_u$.

   k or $\hat{k}$ = 3: Y = Y.

   k = 4: Y_u = same bits as Y_{14}; Y_L = Y.

   k = 5: Y_u = same bits as Y_{14}; $Y_L = (Y)_L$.

   k = 6: Y_u = same bits as Y_{29}; $Y_L = (Y)_u$.

   k = 7: Y = (A).

For instructions 23, 52, and 53, $k = 7$ is not used.

For instruction 13, only $\hat{k} = 3$ is permitted.

For instructions 73 through 76, $\hat{k} = 2$ is not used.

2) *Store* instructions (14 through 16, 17, 32, 33, and 47):

$k = 0$: Store (A or B^j) in Q^*.

$k = 1$: Store (A_L, Q_L, or B^j) in Y_L, leaving $(Y)_u$ undisturbed.

$k = 2$: Store (A_L, Q_L, or B^j) in Y_u, leaving $(Y)_L$ undisturbed.

k or $\hat{k} = 3$: Store (A, Q, C^j, or B^j) in Y.

$k = 4$: Store (Q or B^j) in A^{**}.

$k = 5$: Store complement of (A_L, Q_L, or B^j) in Y_L, leaving $(Y)_u$ undisturbed.

$k = 6$: Store complement of (A_L, Q_L, or B^j) in Y_u, leaving $(Y)_L$ undisturbed.

$k = 7$: Store complement of (A, Q, or B^j) in Y. (Storing the complement of B^j is the same complement as for a 30-bit register.)

For instruction 17, only $\hat{k} = 3$ is permitted.

3) *Replace* instructions (24, 25, 34 through 37, 44 through 46, and 54 through 57):

$k = 0$: Not used.

$k = 1$: *Read* portion - Y_u = 0's; $Y_L = (Y)_L$.

Store portion - stores (A_L, Q_L, or B^j) in Y_L, leaving $(Y)_u$ undisturbed.

$k = 2$: *Read* portion - Y_u = 0's; $Y_L = (Y)_u$.

Store portion - stores (A_L, Q_L, or B^j) in Y_u, leaving $(Y)_u$ undisturbed.

$k = 3$: *Read* portion - Y = Y.

Store portion - stores (A, Q, or B^j) in Y.

$k = 4$: Not used.

$k = 5$: *Read* portion - Y_u = same bits at Y_{14}; $Y_L = (Y)_L$.

Store portion - stores (A_L, Q_L, or B^j) in Y_L, leaving $(Y)_u$ undisturbed.

---

* A 14000 00000 instruction complements (Q).

** A 15040 00000 instruction complements (A).

k = 6: *Read* portion - Y_u = same bits as Y_{29}; $Y_L = Y_u$.

Store portion - stores (A_L, Q_L, or B^j) in Y_u, leaving $(Y)_L$ undisturbed.

k = 7: Not used.

The *Repeat* instruction requires special interpretation when followed by a *Replace* instruction. See details on page 24, Instruction No. 70, *REPEAT*.

F. *INDEX DESIGNATOR* - **b**

The **b** designator (3 bits) appears in bit-positions 17, 16, and 15 of the U-register, or an instruction (see Figure 1), specifying which of the B-registers, if any, will be used to modify the Operand Designator, **y**, to form $Y = y + (B^b)$. This operation employs an additive accumulator; hence, a quantity consisting of all *zeros* cannot result unless the bits of both the Operand Designator, **y**, and (B^b) are all *zeros*.

Effect of the various values of **b**, the Index Designator, is summarized:

b = 0: Do not modify **y**.

b = 1: Add (B^1) to **y** (modulo $2^{15}-1$).

b = 2: Add (B^2) to **y** (modulo $2^{15}-1$).

b = 3: Add (B^3) to **y** (modulo $2^{15}-1$).

b = 4: Add (B^4) to **y** (modulo $2^{15}-1$).

b = 5: Add (B^5) to **y** (modulo $2^{15}-1$).

b = 6: Add (B^6) to **y** (modulo $2^{15}-1$).

b = 7: Add (B^7) to **y** (modulo $2^{15}-1$).

G. *OPERAND DESIGNATOR* - **y**

The **y** designator (15 bits) appears in bit-positions 14 through 0 of an instruction (see Figure 1). The operand or address of the operand, **Y**, is relative to **y** since $Y = y + (B^b)$.

H. *MAGNETIC CORE MEMORY ASSIGNMENT*

The main Magnetic Core memory consists of 32,768 addressable storage locations. Seventy-three of these locations are special-purpose and provide eight distinct functions:

1) The starting address from MASTER CLEAR

2) The Fault Entrance Register

3) The Real-Time Clock Register

4) External Interrupt Entrance Register for each channel

5) Internal Interrupt Entrance Register for each *input* channel

6) Internal Interrupt Entrance Register for each *output* channel

7) Input Buffer Control Register for each *input* channel

8) Output Buffer Control Register for each *output* channel.

Each of the other memory locations are used for:

1) Instruction word storage

2) Data storage.

The following tabulation specifies Magnetic Core Memory Address assignments and associated storage functions.

ADDRESS (octal)	STORAGE FUNCTION
0 0 0 0 0	Initial Starting Address from MASTER CLEAR
0 0 0 0 1	Memory Word
0 0 0 0 2	Memory Word
0 0 0 0 3	Memory Word
0 0 0 0 4	Memory Word
0 0 0 0 5	Memory Word
0 0 0 0 6	Memory Word
0 0 0 0 7	Memory Word
0 0 0 1 0	Memory Word
0 0 0 1 1	Memory Word
0 0 0 1 2	Memory Word
0 0 0 1 3	Memory Word
0 0 0 1 4	Fault Entrance Register
0 0 0 1 5	Memory Word
0 0 0 1 6	Memory Word
0 0 0 1 7	Real-Time Clock Register
0 0 0 2 0	External Interrupt Entrance Register for Channel 0
0 0 0 2 1	External Interrupt Entrance Register for Channel 1
0 0 0 2 2	External Interrupt Entrance Register for Channel 2
0 0 0 2 3	External Interrupt Entrance Register for Channel 3

ADDRESS (octal)	STORAGE FUNCTION
0 0 0 2 4	External Interrupt Entrance Register for Channel 4
0 0 0 2 5	External Interrupt Entrance Register for Channel 5
0 0 0 2 6	External Interrupt Entrance Register for Channel 6
0 0 0 2 7	External Interrupt Entrance Register for Channel 7
0 0 0 3 0	External Interrupt Entrance Register for Channel 8D
0 0 0 3 1	External Interrupt Entrance Register for Channel 9D
0 0 0 3 2	External Interrupt Entrance Register for Channel 10D
0 0 0 3 3	External Interrupt Entrance Register for Channel 11D
0 0 0 3 4	External Interrupt Entrance Register for Channel 12D
0 0 0 3 5	External Interrupt Entrance Register for Channel 13D
0 0 0 3 6	Memory Word
0 0 0 3 7	Memory Word
0 0 0 4 0	Internal Interrupt Entrance Register for Input Channel 0
0 0 0 4 1	Internal Interrupt Entrance Register for Input Channel 1
0 0 0 4 2	Internal Interrupt Entrance Register for Input Channel 2
0 0 0 4 3	Internal Interrupt Entrance Register for Input Channel 3
0 0 0 4 4	Internal Interrupt Entrance Register for Input Channel 4
0 0 0 4 5	Internal Interrupt Entrance Register for Input Channel 5
0 0 0 4 6	Internal Interrupt Entrance Register for Input Channel 6
0 0 0 4 7	Internal Interrupt Entrance Register for Input Channel 7
0 0 0 5 0	Internal Interrupt Entrance Register for Input Channel 8D
0 0 0 5 1	Internal Interrupt Entrance Register for Input Channel 9D
0 0 0 5 2	Internal Interrupt Entrance Register for Input Channel 10D
0 0 0 5 3	Internal Interrupt Entrance Register for Input Channel 11D
0 0 0 5 4	Internal Interrupt Entrance Register for Input Channel 12D
0 0 0 5 5	Internal Interrupt Entrance Register for Input Channel 13D
0 0 0 5 6	Memory Word
0 0 0 5 7	Memory Word
0 0 0 6 0	Internal Interrupt Entrance Register for Output Channel 0
0 0 0 6 1	Internal Interrupt Entrance Register for Output Channel 1
0 0 0 6 2	Internal Interrupt Entrance Register for Output Channel 2
0 0 0 6 3	Internal Interrupt Entrance Register for Output Channel 3
0 0 0 6 4	Internal Interrupt Entrance Register for Output Channel 4
0 0 0 6 5	Internal Interrupt Entrance Register for Output Channel 5

ADDRESS (octal)	STORAGE FUNCTION
00066	Internal Interrupt Entrance Register for Output Channel 6
00067	Internal Interrupt Entrance Register for Output Channel 7
00070	Internal Interrupt Entrance Register for Output Channel 8D
00071	Internal Interrupt Entrance Register for Output Channel 9D
00072	Internal Interrupt Entrance Register for Output Channel 10D
00073	Internal Interrupt Entrance Register for Output Channel 11D
00074	Internal Interrupt Entrance Register for Output Channel 12D
00075	Internal Interrupt Entrance Register for Output Channel 13D
00076	Memory Word
00077	Memory Word
00100	Input Buffer Control Register for Input Channel 0
00101	Input Buffer Control Register for Input Channel 1
00102	Input Buffer Control Register for Input Channel 2
00103	Input Buffer Control Register for Input Channel 3
00104	Input Buffer Control Register for Input Channel 4
00105	Input Buffer Control Register for Input Channel 5
00106	Input Buffer Control Register for Input Channel 6
00107	Input Buffer Control Register for Input Channel 7
00110	Input Buffer Control Register for Input Channel 8D
00111	Input Buffer Control Register for Input Channel 9D
00112	Input Buffer Control Register for Input Channel 10D
00113	Input Buffer Control Register for Input Channel 11D
00114	Input Buffer Control Register for Input Channel 12D
00115	Input Buffer Control Register for Input Channel 13D
00116	Memory Word
00117	Memory Word
00120	Output Buffer Control Register for Output Channel 0
00121	Output Buffer Control Register for Output Channel 1
00122	Output Buffer Control Register for Output Channel 2
00123	Output Buffer Control Register for Output Channel 3
00124	Output Buffer Control Register for Output Channel 4
00125	Output Buffer Control Register for Output Channel 5
00126	Output Buffer Control Register for Output Channel 6
00127	Output Buffer Control Register for Output Channel 7

ADDRESS (octal)	STORAGE FUNCTION
0 0 1 3 0	Output Buffer Control Register for Output Channel 8D
0 0 1 3 1	Output Buffer Control Register for Output Channel 9D
0 0 1 3 2	Output Buffer Control Register for Output Channel 10D
0 0 1 3 3	Output Buffer Control Register for Output Channel 11D
0 0 1 3 4	Output Buffer Control Register for Output Channel 12D
0 0 1 3 5	Output Buffer Control Register for Output Channel 13D

(0 0 1 3 6 - 0 7 7 7 7) = 4,002D words of memory

(1 0 0 0 0 - 1 7 7 7 7) = 4,096D words of memory

(2 0 0 0 0 - 2 7 7 7 7) = 4,096D words of memory

(3 0 0 0 0 - 3 7 7 7 7) = 4,096D words of memory

(4 0 0 0 0 - 4 7 7 7 7) = 4,096D words of memory

(5 0 0 0 0 - 5 7 7 7 7) = 4,096D words of memory

(6 0 0 0 0 - 6 7 7 7 7) = 4,096D words of memory

(7 0 0 0 0 - 7 7 7 7 7) = 4,096D words of memory

I. *WIRED MEMORY* - The AN/USQ-20 Unit Computer contains 16D words of semipermanent wired core storage. Programming this memory area requires a process of *wiring-in* the desired instructions. The semipermanent feature of these storage locations prevents accidental destruction of program instructions contained therein since entries cannot be made via main memory.

An Input Bootstrap routine occupies this memory, and its execution is controlled by the Automatic Recovery Switch.

J. *AUTOMATIC RECOVERY* - In the event of a *fault* condition (encountering either a 00 or 77 function code), the Automatic Recovery Switch directs computer activity. This switch has three positions: 1) DOWN, 2) NEUTRAL, and 3) UP. Action resulting from these positions is:

1) DOWN position - This causes manual execution of the Bootstrap routine. Computer action begins at address 0 of Wired Memory and executes the Bootstrap routine when this switch is depressed. (A MASTER CLEAR should precede this operation.)

2) NEUTRAL position - This causes an Interrupt to address 00014 of Main Memory on a fault condition. Action continues as programmed.

3) UP position - This causes an Interrupt to address 14 of Wired Memory on a fault condition. This results in automatic execution of the Bootstrap routine.

K. *BUFFER MODES* - The AN/USQ-20 Unit Computer provides two modes of buffering: 1) with monitor and 2) without monitor.

Buffering with monitor transfers words sequentially, starting at a given initial address through a given terminal address, on the specified input or output channel. The computer continues execution of program instructions during the buffer process. Completion of the buffering process causes an Internal Monitor Interrupt to the Internal Interrupt Entrance Register assigned to the input or output channel. (See subsection H, *MAGNETIC CORE MEMORY ASSIGNMENT*.) This register should contain a RETURN JUMP instruction*. (See Instructions 75 and 76.)

Buffering without the monitor transfers words sequentially, starting at a given initial address through a given terminal address, on a specified input or output channel. The computer continues execution of program instructions during the buffer process. No monitor interrupt will occur. (See Instructions 73 and 74.)

3. LIST OF INSTRUCTIONS

This section lists the repertoire of instructions used with the NTDS AN/USQ-20 Unit Computer. Common usage of these instructions is also included; no attempt is made to indicate more sophisticated use.

01 *RIGHT SHIFT Q*

This instruction shifts (Q) to the right **Y** bit positions. The higher-order bits are replaced with the original sign bit as the word is shifted. Only the lower-order six bits of **Y** are recognized for this instruction. The higher-order 24 bits are ignored.

Example of right shift in Q: **Y** = 2

Content of Q		Content of Q	
$(Q)_i$ (positive) =	0 1 0 1	$(Q)_i$ (negative) =	1 0 1 0
First shift	0 0 1 0	First shift	1 1 0 1
Second shift	0 0 0 1	Second shift	1 1 1 0

02 *RIGHT SHIFT A*

This instruction shifts (A) to the right **Y** bit positions. The higher-order bits are replaced with the original sign bit as the word is shifted. Only the lower-order six bits of

* Suggested instruction for the Internal Interrupt Register is:
650nn nnnnn - Exit to an Interrupt subroutine for remedial action. This subroutine ends with a 601nn instruction which clears the Interrupt mode, then returns control to the main routine.

Y are recognized for this instruction. The higher-order 24 bits are ignored. The overall operation is analogous to the example given in the foregoing instruction.

03 *RIGHT SHIFT AQ*

This instruction shifts (A) and (Q) as one 60-bit register. The shift is to the right Y bit positions with the lower-order bits of A shifting into the higher-order bit positions of Q. The higher-order bits of A are replaced with the original sign bit as the word is shifted. Only the lower-order six bits of Y are recognized for this instruction. The higher-order 24 bits are ignored.

Example of right shift in AQ: Y = 2

Content of AQ		Content of AQ	
$(AQ)_i$ (positive) =	0 1 0 1 0 0 1 1	$(AQ)_i$ (negative) =	1 0 0 0 1 0 1 0
First shift	0 0 1 0 1 0 0 1	First shift	1 1 0 0 0 1 0 1
Second shift	0 0 0 1 0 1 0 0	Second shift	1 1 1 0 0 0 1 0

04 *COMPARE*

This instruction compares the signed value of Y with the signed value of (A) and/or (Q). It does not alter either (A) or (Q). The Branch Condition Designator, j, is interpreted in a special way for this instruction as listed below:

j = 0: Do not skip the next instruction.

j = 1: Skip the next instruction.

j = 2: Skip the next instruction if Y is less than, or equal to, (Q).

j = 3: Skip the next instruction if Y is greater than (Q).

j = 4: Skip the next instruction if (Q) is greater than, or equal to Y, and Y is greater than (A).

j = 5: Skip the next instruction if Y is greater than (Q) or if Y is less than, or equal to, (A).

j = 6: Skip the next instruction if Y is less than, or equal to, (A).

j = 7: Skip the next instruction if Y is greater than (A).

05 *LEFT SHIFT Q*

This instruction shifts (Q) circularly to the left Y bit positions*. The lower-order bits

* Maximum shift count permitted is 59D places.

are replaced with the higher-order bits as the word is shifted. Only the lower-order six bits of Y are recognized for this instruction. The higher-order 24 bits are ignored.

Example of left circular shift in Q: (Y) = 2

Content of Q		Content of Q	
$(Q)_i$ (positive) =	0 0 1 1	$(Q)_i$ (negative) =	1 1 0 0
First shift	0 1 1 0	First shift	1 0 0 1
Second shift	1 1 0 0	Second shift	0 0 1 1

06 *LEFT SHIFT A*

This instruction shifts (A) circularly to the left Y bit positions.* The lower-order bits are replaced with the higher-order bits as the word is shifted. Only the lower-order six bits of Y are recognized for this instruction. The higher-order 24 bits are ignored. The over-all operation is analogous to the example given in the foregoing instruction.

07 *LEFT SHIFT AQ*

This instruction shifts (A) and (Q) as one 60-bit register. The shift is circular to the left Y bit positions.* The lower-order bits of A are replaced with the higher-order bits of Q and the lower-order bits of Q are replaced with the higher-order bits of A. Only the lower-order six bits of Y are recognized by this instruction. The higher-order 24 bits are ignored.

Example of left circular shift in AQ: Y = 2

Content of AQ		Content of AQ	
$(AQ)_i$ (positive) =	0 1 0 1 0 0 1 1	$(AQ)_i$ (negative) =	1 0 0 0 1 0 1 1
First shift	1 0 1 0 0 1 1 0	First shift	0 0 0 1 0 1 1 1
Second shift	0 1 0 0 1 1 0 1	Second shift	0 0 1 0 1 1 1 0

10 *ENTER Q*

Clear the Q-register. Then transmit Y to Q.

11 *ENTER A*

Clear A. Then transmit Y to A.

12 *ENTER* B^n

Clear B-register j. Then transmit the lower-order 15 bits of Y to B-register j. The higher-order 15 bits of Y are ignored in this instruction. The Branch Condition Desig-

* Maximum shift count permitted is 59D places.

nator, j , is used to specify the selected B-register for this instruction and is not available for its normal function.

13 *EXTERNAL FUNCTION ON* C^n

$\hat{j}$ = 0 or 1. Interrogate the two bits connected to the input-active designator (flip-flops) on an interconnected computer. If the interconnected computer's input buffer is active, skip the next instruction. If the interconnected computer's input buffer is not active, execute the next instruction. There are no External Function lines on C^0 or C^1. $\hat{k}$ = 3 is required for timing when $\hat{j} \neq 0$ or 1. Transmit Y, the External Function, over the channel specified by $\hat{j}$. Only $\hat{k}$ = 3 is permitted.

14 *STORE Q*

Store (Q) at storage address Y as directed by the Operand Interpretation Designator, k. If k = 0, complement (Q). If k = 4, store in A.

15 *STORE A*

Store (A) at storage address Y as directed by the Operand Interpretation Designator, k. If k = 4, complement (A). If k = 0, store in Q.

16 *STORE* B^n

Store a 30-bit quantity whose lower-order 15 bits correspond to the content of B-register j and whose higher-order 15 bits are *zero* at storage address Y as directed by the Operand Interpretation Designator, k. The Branch Condition Designator, j, is used to specify the selected B-register for this instruction and is not available for its normal function.

17 *STORE* C^n

Store the content of the C-channel specified by $\hat{j}$ at storage address Y. An Input Acknowledge signal is then sent on the C-channel. Only $\hat{k}$ = 3 is permitted.

20 *ADD A*

Add Y to the previous content of the Accumulator.

21 *SUBTRACT A*

Subtract Y from the previous content of the Accumulator.

22 *MULTIPLY*

Multiply (Q) times Y leaving the double-length product in AQ. If the factors are considered as integers, the product is an integer in AQ.

The Branch Condition Designator, j, is interpreted prior to end correction permitting sensing of a product with $(A)_f = 0$. If j equal 4, a skip of the next instruction is made when $(A)_f = 0$. When $(A)_f \neq +0$, a double-length product has been formed with significant bit(s) in the Accumulator; however, if a Skip does occur for j = 4, the Multiply instruction can be re-executed with the same operand and with j = 2 or 3 to determine if Q_{29} contains a significant bit (a *one*) of the product.

In this instruction, k = 7 should not be used.

23 *DIVIDE*

Divide (AQ) by Y leaving the quotient in the Q-register and the remainder in the A-register. The remainder bears the same sign as the quotient. In this instruction, k = 7 should not be used.

NOTE:

If a DIVIDE FAULT condition exists, no Maintenance Console indication is given; however, by coding each Divide instruction with j = 3, a program test for the DIVIDE FAULT is automatic. With this selection of j, a Skip of the next instruction occurs if a DIVIDE FAULT exists. The Skip should be made to a Jump instruction which provides a remedial means of noting or correcting the error. Therefore, the instruction which follows the Divide instruction should have its j = 1 in order to preclude the Jump instruction whenever the "Divide Sequence" culminates in a correct answer.
A DIVIDE FAULT can also be detected if the Divide instruction is executed with j = 2. In this case, a correct answer is indicated when a Skip occurs.

24 *REPLACE A + Y*

Add (Y) to the previous content of A. Store (A) at storage address Y.

25 *REPLACE A - Y*

Subtract (Y) from the previous content of A. Then store (A) at storage address Y.

26 *ADD Q*

Interchange (A) and (Q). Then add Y to (A). Interchange (A) and (Q). The content of A is undisturbed by this instruction. The Branch Condition Designator, j, has special meaning in this instruction as in instruction 27.

27 *SUBTRACT Q*

Interchange (A) and (Q). Then subtract Y from (A). Interchange (A) and (Q). The con-

tent of A is undisturbed by this instruction. The Branch Condition Designator, j, has special meaning in this instruction as listed below.

NOTE:

In instructions 26 and 27 the Branch Condition Designator, j, has the following meaning:

- j = 0: *Do not skip the next instruction.*
- j = 1: *Skip the next instruction.*
- j = 2: *Skip the next instruction if (A) is positive.*
- j = 3: *Skip the next instruction if (A) is negative.*
- j = 4: *Skip the next instruction if (Q) is zero.*
- j = 5: *Skip the next instruction if (Q) is nonzero.*
- j = 6: *Skip the next instruction if (Q) is positive.*
- j = 7: *Skip the next instruction if (Q) is negative.*

30 *ENTER Y + Q*

Clear A. Then transmit (Q) to A. Then add Y to (A).

31 *ENTER Y - Q*

Clear A. Then transmit (Q) to A. Then subtract Y from (A). Finally, complement (A).

32 *STORE A + Q*

Add (Q) to the previous content of A. Then store (A) at storage address Y as directed by the Operand Interpretation Designator, k.

33 *STORE A - Q*

Subtract (Q) from the previous content of A. Then store (A) at storage address Y as directed by the Operand Interpretation Designator, k.

34 *REPLACE Y + Q*

Clear A. Then transmit (Q) to A. Then add (Y) to (A). Then store (A) at storage address Y.

35 *REPLACE Y - Q*

Clear A. Then transmit (Q) to A. Then subtract (Y) from (A). Then complement (A) and store at storage address Y.

36 *REPLACE Y + 1*

Clear A. Then set (A) = 1. Then add (Y) to (A). Then store (A) at storage address Y.

37 *REPLACE Y - 1*

Clear A. Then set (A) = 1. Then subtract (Y) from (A). Then complement (A) and store at storage address Y.

40 *ENTER LOGICAL PRODUCT*

Enter in A the bit-by-bit product of **Y** and (Q).

The **j** designator is interpreted in a special way for this instruction for the value **j** = 2 or 3. If **j** = 2, Skip if the parity of $(A)_f$ is even. If **j** = 3, Skip if the parity of $(A)_f$ is odd.

NOTE:

Even parity = an even number of "ones" in the A-register.
Odd parity = an odd number of "ones" in the A-register.

41 *ADD LOGICAL PRODUCT*

Add to (A) the bit-by-bit product of **Y** and (Q).

42 *SUBTRACT LOGICAL PRODUCT*

Subtract from (A) the bit-by-bit product of **Y** and (Q).

43 *COMPARE MASKED*

Subtract from (A) the bit-by-bit product of **Y** and (Q), and perform the branch point evaluation for Skip of next sequential instruction as directed by the Branch Condition Designator, **j**.

This instruction results in no net change in the content of any operational register. It provides, through the Branch Condition Designator, **j**, a comparison of a portion of **Y** with (A).

44 *REPLACE LOGICAL PRODUCT*

Enter in A the bit-by-bit product of (Y) and (Q). Then store (A) at storage address Y.

The **j** designator is interpreted in a special way for this instruction for the values **j** = 2 or 3. If **j** = 2, Skip if the parity of $(A)_f$ is even. If **j** = 3, Skip if the parity of $(A)_f$ is odd.

NOTE:

Even parity = an even number of "ones" in the A-register.
Odd parity = an odd number of "ones" in the A-register.

45 *REPLACE A + LOGICAL PRODUCT*

Add to (A) the bit-by-bit product of (Y) and (Q). Then store (A) at storage address Y.

46 *REPLACE A - LOGICAL PRODUCT*

Subtract from (A) the bit-by-bit product of (Y) and (Q). Then store (A) at storage address Y.

47 *STORE LOGICAL PRODUCT*

Store in address Y the bit-by-bit product of (A) and (Q) as directed by the Operand Interpretation Designator, **k**.

50 *SELECTIVE SET*

Set the individual bits of A to *one* corresponding to *ones* in **Y** leaving the remaining bits of A unaltered.

51 *SELECTIVE COMPLEMENT*

Complement the individual bits of A corresponding to *ones* in **Y** leaving the remaining bits of A unaltered.

52 *SELECTIVE CLEAR*

Clear the individual bits of A corresponding to *ones* in **Y** leaving the remaining bits of A unaltered.

In this instruction, **k** = 7 should not be used.

53 *SELECTIVE SUBSTITUTE*

Set the individual bits of A with bits of **Y** corresponding to *ones* in Q leaving the remaining bits of A unaltered.

In this instruction, **k** = 7 should not be used.

54 *REPLACE SELECTIVE SET*

Set the individual bits of A to *one* corresponding to *ones* in (Y) leaving the remaining bits of A unaltered. Then store (A) at storage address Y.

55 *REPLACE SELECTIVE COMPLEMENT*

Complement the individual bits of A corresponding to *ones* in (Y) leaving the remaining bits of A unaltered. Then store (A) at storage address Y.

56 *REPLACE SELECTIVE CLEAR*

Clear individual bits of A corresponding to *ones* in (Y) leaving the remaining bits of A unaltered. Then store (A) at storage address Y.

57 *REPLACE SELECTIVE SUBSTITUTE*

Clear individual bits of A corresponding to *ones* in Q leaving the remaining bits of A unaltered. Then form the bit-by-bit product of (Y) and (Q), and set *ones* of this product in corresponding bits of A leaving the remaining bits of A unaltered. Then store (A) at storage address Y.

60 *JUMP (Arithmetic)*

This instruction clears the Program Address Register, P, and enters a new program address in P for certain conditions of either the A- or Q-register content. The Branch Condition Designator, j, is interpreted in a special way for this instruction and thus determines the conditions under which a Jump in program address occurs. If the Jump condition is not satisfied, the next sequential instruction in the current sequence is executed in a normal manner. If the Jump condition is satisfied, as listed below, then Y becomes the address of the next instruction and the beginning of a new program sequence.

- j = 0: No jump. Set Interrupt Enable to remove interrupt lockout, thus clearing Bootstrap and Interrupt modes. Continue with current program sequence.
- j = 1: Execute jump. Set Interrupt Enable to remove interrupt lockout, thus clearing Bootstrap and Interrupt modes.
- j = 2: Execute jump if (Q) is positive.
- j = 3: Execute jump if (Q) is negative.
- j = 4: Execute jump if (A) is zero.
- j = 5: Execute jump is (A) is nonzero.
- j = 6: Execute jump if (A) is positive.
- j = 7: Execute jump if (A) is negative.

61 *JUMP (Manual)*

This instruction clears the Program Address Register, P, and enters a new program address in P for certain conditions of manual JUMP key selections. The Branch Condition Designator, j, is interpreted in a special way for this instruction and thus determines the conditions under which a jump in program address occurs. If the Jump condition is not satisfied, the next sequential instruction in the current sequence is executed

in a normal manner. If the Jump condition is satisfied, as listed below, then Y becomes the address of the next instruction and the beginning of a new program sequence.

Program execution may be stopped by certain STOP selections on execution of this instruction. The Branch Condition Designator, j, specifies which key selections are effective.

- j = 0: Execute jump regardless of key selections.
- j = 1: Execute jump if JUMP 1 is selected.
- j = 2: Execute jump if JUMP 2 is selected.
- j = 3: Execute jump if JUMP 3 is selected.
- j = 4: Execute jump. Stop computation.
- j = 5: Execute jump. Stop computation if STOP 5 is selected.
- j = 6: Execute jump. Stop computation if STOP 6 is selected.
- j = 7: Execute jump. Stop computation if STOP 7 is selected.

62 *JUMP ON C^n ACTIVE INPUT BUFFER*

This instruction clears the Program Address Register, P, and enters a new program address in P for certain input buffer conditions on the channel designated by $\hat{j}$. If the buffer is active, the Jump condition is satisfied; then Y becomes the address of the next instruction. If the buffer in inactive, the Jump condition is not satisfied. The next sequential instruction in the current sequence is executed in the normal manner. $\hat{k}$ = 0, 1, 2, or 3 is permitted.

63 *JUMP ON C^n ACTIVE OUTPUT BUFFER*

This instruction clears the Program Address Register, P, and enters a new address in P for certain output buffer conditions on the channel designated by $\hat{j}$. If the buffer is active, the Jump condition is satisfied; then Y becomes the address of the next instruction. If the buffer is inactive, the Jump condition is not satisfied. The next sequential instruction in the current sequence is executed in the normal manner. $\hat{k}$ = 0, 1, 2, or 3 is permitted.

64 *RETURN JUMP (Arithmetic)*

This instruction executes a Return Jump sequence for certain conditions of either the A- or Q-register content. The Branch Condition Designator, j, is interpreted in a special way for this instruction and determines the conditions under which the Return Jump sequence is executed. If the Return Jump condition is not satisfied, the next sequential instruction in the current sequence is executed in a normal manner. If the Return Jump condition is satisfied, as listed below, the following sequence is performed.

Store (P) + 1 in the lower half of memory address Y. Then jump to Y + 1.

- j = 0: No action; continue with the current program sequence.
- j = 1: Execute return jump.
- j = 2: Execute return jump if (Q) is positive.
- j = 3: Execute return jump if (Q) is negative.
- j = 4: Execute return jump if (A) is zero.
- j = 5: Execute return jump if (A) is nonzero.
- j = 6: Execute return jump if (A) is positive.
- j = 7: Execute return jump if (A) is negative.

65 *RETURN JUMP (Manual)*

This instruction executes a Return Jump sequence for certain conditions of manual key selections. The Branch Condition Designator, j, is interpreted in a special way for this instruction and determines the conditions under which the Return Jump sequence is executed. If the Return Jump condition is not satisfied, the next sequential instruction in the current sequence is executed in a normal manner. If the Return Jump condition is satisfied, as listed below, the following sequence is performed.

Store (P) + 1 in the lower half of memory address Y. Then jump to Y + 1.

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

66 *TERMINATE C^n INPUT BUFFER*

This instruction terminates the input buffer on channel $\hat{j}$ No Input Buffer Monitor Interrupt will occur.

The Operand Interpretation Designator, $\hat{k}$, the Index Designator, b, and the Operand Designator, Y, bits are not translated for this instruction.

67 *TERMINATE C^n OUTPUT BUFFER*

This instruction terminates the output buffer on channel $\hat{j}$. No Output Buffer Monitor Interrupt will occur.

The Operand Interpretation Designator, $\hat{k}$, the Index Designator, **b**, and the Operand Designator, **Y**, bits are not translated for this instruction.

70 *REPEAT*

Clear B^7 and transmit the lower 15 bits of **Y** to B^7. If **Y** is nonzero, transmit (j) to **r** (designator register), thereby, initiating the *repeat mode*. If **Y** is zero, skip the next instruction.

REPEAT MODE - The *repeat mode* executes the instruction immediately following the Repeat instruction **Y** times; B^7 contains the number of executions remaining throughout the repeat mode.

If no Skip condition is met for the repeated instruction, the *repeat mode* terminates. The instruction following the repeated instruction is then executed. If the Skip condition for the repeated instruction is met, the *repeat mode* terminates, and the instruction following the repeated instruction is skipped.

Following the *repeat mode* termination, the count remains in B^7. In no way does the *repeat mode* alter a repeated instruction as stored in memory.

The three low-order bits of the **r** designator (from **j** of instruction 70) affect operand indexing as follows:

r = 0: Do not modify the operand address of the repeated instruction after each individual execution.

r = 1: Increase the operand address of the repeated instruction by one after each execution of the repeated instruction.

r = 2: Decrease the operand address of the repeated instruction by one after each execution of the repeated instruction.

r = 3: Repeat the initial B-register modification of the repeated instruction before each execution.

r = 4: Do not modify the operand address of the repeated instruction after each individual execution. If the repeated instruction is a Replace instruction, the operand address is incremented by (B^6) for the store portion of the Replace Instruction.

r = 5: Increase the operand address of the repeated instruction by one after each execution of the repeated instruction. If the repeated instruction

is a Replace instruction, the operand address is incremented by (B^6) for the store portion of the Replace instruction.

r = 6: Decrease the operand address of the repeated instruction by one after each execution of the repeated instruction. If the repeated instruction is a Replace instruction, the operand address is incremented by (B^6) for the store portion of the Replace instruction.

r = 7: Repeat the initial B-register modification of the repeated instruction before each execution. If the repeated instruction is a Replace instruction, the operand address is incremented by (B^6) for the store portion of the Replace instruction.

NOTE:

Instruction 70 j *designator establishes the repeat mode* r *designator since* j *is transmitted to* r.

71 *B SKIP ON* B^n

If the content of B-register j is equal to Y, skip the next instruction in the current sequence and proceed to the instruction following. Clear B-register j.

If the content of B-register j is not equal to Y, proceed to the next instruction in the sequence in a normal manner. Increase the content of B-register j by one.

The Branch Condition Designator, j, is used to designate the selected B-register in this instruction and is not available for its normal function. Only the lower-order 15 bits of Y are used in the comparison described in the preceding paragraph.

72 *B JUMP ON* B^n

If the content of B-register j is *nonzero* execute a jump in program address to address Y. Reduce the content of B-register j by one.

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

The Branch Condition Designator, j, is used to designate the selected B-register in this instruction and is not available for its normal function. If the Jump condition is satisfied, then the lower-order 15 bits of Y become the address of the next instruction and the beginning of the new program sequence. The higher-order 15 bits of (Y) cannot be used in this instruction.

73 *INPUT BUFFER ON C^n (without MONITOR Mode)*

This instruction establishes an input buffer via input buffer channel $\hat{j}$ to Magnetic Core Storage with an initial storage address Y. Subsequent to this instruction, individual transfers will be executed at a rate determined by an external device. The storage address initially established by this instruction will be advanced by one preceding each individual transfer. The next current address will be maintained throughout the buffer process in the lower-order 15 bits of Magnetic Core Storage address 00100 plus $\hat{j}$. This mode will continue until it is superseded by a subsequent initiation or termination of an input buffer via the same input channel or until the higher-order half and the lower-order half of storage address 00100 plus $\hat{j}$ contain equal quantities, whichever occurs first.

This instruction is implemented as follows: If $\hat{k} = 3$, store (Y) in storage location 00100 plus $\hat{j}$. If $\hat{k} = 1$, store the lower-order 15 bits of (Y) in the lower-order half of storage location 00100 plus $\hat{j}$ leaving the higher-order half undisturbed. If $\hat{k} = 0$, store Y in the lower-order half of storage location 00100 plus $\hat{j}$ leaving the higher-order half undisturbed. Proceed to the next instruction. $\hat{k} = 2$ is not permitted.

74 *OUTPUT BUFFER ON C^n (without MONITOR Mode)*

This instruction establishes an output buffer via output buffer channel $\hat{j}$ from initial storage address Y in Magnetic Core Storage. Subsequent to this instruction, the individual transfers will be executed at a rate determined by an external device. The storage address initially established by this instruction will be advanced by one preceding each individual transfer. The next current address will be maintained throughout the buffer process in the lower-order 15 bits of Magnetic Core Storage address 00120 plus $\hat{j}$. This mode will continue until it is superseded by a subsequent initiation or termination of an output buffer via the same output channel or until the higher-order half and the lower-order half of storage address 00120 plus $\hat{j}$ contain equal quantities, whichever occurs first.

This instruction is implemented as follows: If $\hat{k} = 3$, store (Y) in storage location 00120 plus $\hat{j}$. If $\hat{k} = 1$, store the lower-order 15 bits of (Y) in the lower-order half of storage location 00120 plus $\hat{j}$ leaving the higher-order half undisturbed. If $\hat{k} = 0$, store Y in the lower-order half of storage location 00120 plus $\hat{j}$ leaving the higher-order half undisturbed. Proceed to the next instruction. $\hat{k} = 2$ is not permitted.

75 *INPUT BUFFER ON C^n (with MONITOR Mode)*

This instruction establishes an input buffer via input buffer channel $\hat{j}$ to Magnetic Core Storage with an initial storage address Y. Subsequent to this instruction, the individual

transfers will be executed at a rate determined by an external device. The storage address initially established by this instruction will be advanced by one preceding each individual transfer. The next current address will be maintained throughout the buffer process in the lower-order 15 bits of Magnetic Core Storage address 00100 plus $\hat{j}$. This mode will continue until it is superseded by a subsequent initiation or termination of an input buffer via the same input channel or until the higher-order half and the lower-order half of storage address 00100 plus $\hat{j}$ contain equal quantities, whichever occurs first. Initiation of this input buffer selects the input channel $\hat{j}$ and establishes a buffer monitor on input channel $\hat{j}$. A Monitor Interrupt follows completion of the buffering operation: $(00100 + j)_u = (00100 + j)_L$.

This instruction is implemented as follows: If $\hat{k} = 3$, store (Y) in storage location 00100 plus $\hat{j}$. If $\hat{k} = 1$, store the lower-order 15 bits of (Y) in the lower-order half of storage location 00100 plus $\hat{j}$ leaving the higher-order half undisturbed. If $\hat{k} = 0$, store Y in the lower-order half of storage location 00100 plus $\hat{j}$. Proceed to the next instruction. $\hat{k} = 2$ is not permitted.

76 *OUTPUT BUFFER ON C^n (with MONITOR Mode)*

This instruction establishes an output buffer via output buffer channel $\hat{j}$ from initial storage address Y in Magnetic Core Storage. Subsequent to this instruction, the individual transfers will be executed at a rate determined by an external device. The storage initially established by this instruction will be advanced by one preceding each individual transfer. The next current address will be maintained throughout the buffer process in the lower-order 15 bits of Magnetic Core Storage address 00120 plus $\hat{j}$. This mode will continue until it is superseded by a subsequent initiation or termination of an output buffer via the same output channel or until the higher-order half and the lower-order half of storage address 00120 plus $\hat{j}$ contain equal quantities, whichever occurs first. Initiation of this output buffer selects the output channel $\hat{j}$ and establishes a buffer monitor on output channel $\hat{j}$. A Monitor Interrupt follows the completion of the buffering operation: $(00120 + j)_u = (00120 + j)_L$.

This instruction is implemented as follows: If $\hat{k} = 3$, store (Y) in storage location 00120 plus $\hat{j}$. If $\hat{k} = 1$, store the lower-order 15 bits of (Y) in the lower-order half of storage location 00120 plus $\hat{j}$ leaving the higher-order half undisturbed. If $\hat{k} = 0$, store Y in the lower-order half of storage location 00120 plus $\hat{j}$ leaving the higher-order half undisturbed. Proceed to the next instruction. $\hat{k} = 2$ is not permitted.

TECHNICAL NOTE NO. 240

DISTRIBUTION LIST

BuShips Code 687E	(8)
NEL Code 1800	(20)
NEL Code 2800	(6)
St. Paul Central File	(250)
San Diego Central File	(50)
A. P. Hendrickson	
G. G. Chapin	
C. W. Glewwe	
R. A. Hileman	
C. J. Homan	
M.M. Koschmann	
G. E. Pickering	
J. A. Kershaw	
F. E. McLeod	
R. P. Fischer	
H. K. Smead	
T. O. Robinson	(2)
C. J. Haggerty	(2)
Contracts Department	(2)
Bureau of Ships Technical Representative - St. Paul	
W. G. Haberstroh	
E. G. Runyon	
R. L. Burkholder	
G. R. Kregness	
H. D. Wise	

Approved: F. E. McLeod
F. E. McLeod
Asst. Department Manager
Computer Design

Approved: G. G. Chapin
G. G. Chapin
Asst. Department Manager
Systems Development

Approved: A. P. Hendrickson
A. P. Hendrickson
Manager
Naval Tactical Data System

NTDS UNIT COMPUTER

AN/USQ-20

Repertoire of Instructions

JP & RJP j-DESIGNATORS

j	JP 60 64	RJP 60 64	JP 61 65	RJP 61 65
0	(No Jump)*		(Uncond. Jump)	
1	(Uncond. Jump)*		KEY 1	
2	Q POS		KEY 2	
3	Q NEG		KEY 3	
4	A ZERO		STOP	
5	A NOT Zero		STOP 5	
6	A POS		STOP 6	
7	A NEG		STOP 7	
	62 ↑		63 ↑	
0-5	C^n ACTIVE IN		C^n ACTIVE OUT	

*60 Clears interrupt & bootstrap modes.

↑j-DESIGNATORS

(4 bits)

↑ Occupies 4 bit positions and represents C^n where n may be $0-15_8$

The instruction word assumes the format:

f		h	y
29 – 24	23 – 20	19 17 – 15	14 – 0

↑k-DESIGNATORS

(2 bits)

k	EX–FCT 13	STR+C^n 17	JP 62 63	IN+C^n; OUT+C^n 73 75 76 76
0	'not used'	'not used'	'blank'	'blank'
1	'not used'	'not used'	L	L
2	'not used'	'not used'	U	'not used'
3	W	W	W	W

*j-DESIGNATORS

j	COM=A, +Q, +AQ 04	DIV 23	ADD=Q, SUB=Q 26 27	EXT=LP, RPL=LP 40 44	RPT 70	
0	(no skip)	(no skip)	(no skip)	(no skip)	(no mod.)	Y of NE + Y	
1	(unconditional skip)	SKIP	SKIP	SKIP	ADV	Y of NE + Y+1	
2	Y LESS | Y ≤ (Q)	NO Over Flow	A POS	EVEN parity	BACK	Y of NE + Y−1	
3	Y MORE | Y > (Q)	Over Flow	A NEG	ODD parity	ADD B	Y of NE + Y + B^5	
4	Y IN (Q)≥ Y and Y >(A)	A ZERO	Q ZERO	A ZERO	Rpt. Inc.	Y of NE+Y | + B^5	✓
5	Y OUT (Q)< Y or Y ≤(A)	A NOT Zero	Q NOT Zero	A NOT Zero	ADV R*	Y of NE+Y+1 | + B^5	✓
6	Y LESS | Y ≤ (A)	A POS	Q POS	A POS	BACK R	Y of NE + Y−1 | + B^5	✓
7	Y MORE | Y > (A)	A NEG	Q NEG	A NEG	ADD B R	Y of NE+Y+B^5 | + B^5	✓

✓ B^5 increment if N5 is RPL class; increments Y address for the store portion of the replace.
NE – Next execution

NORMAL j-DESIG.

j	(Not applicable on + or −) Skip Code
0	(no skip)
1	SKIP
2	Q POS
3	Q NEG
4	A ZERO
5	A NOT Zero
6	A POS
7	A NEG

NORMAL k-DESIGNATORS

k	READ Code	READ Origin	STORE Code	STORE Dest.	REPLACE Code	REPLACE Origin	REPLACE Dest.
0	'blank'	U_L	Q	Q	'not used'	—	—
1	L	M_L	L	M_L	L	M_L	M_L
2	U	M_U	U	M_U	U	M_U	M_U
3	W	M	W	M	W	M	M
4	X	XU_L	A	A	'not used'	—	—
5	LX	XM_L	CPL	Cpl M_L	LX	XM_L	M_L
6	UX	XM_U	CPU	Cpl M_U	UX	XM_U	M_U
7	A	A	CPW	Cpl M	'not used'	—	—

LEGEND.

M – Memory word (30 bits)
M_L – Lower half memory word
M_U – Upper half memory word
X – Sign bit extended
Cpl – Complement
A – A-register
Q – Q-register
U – U-register

NTDS UNIT COMPUTER

AN/USQ-20 — *Repertoire of Instructions*

01	Right SHift • Q	Shift (Q) Right by Y
02	Right SHift • A	Shift (A) Right by Y
03	Right SHift • AQ	Shift (AQ) Right by Y
04*	COMpare • A, • Q, • AQ.......	Sense (j); (A)$_j$ = (A)$_f$
05	Left SHift • Q...........	Shift (Q) Left by Y
06	Left SHift • A	Shift (A) Left by Y
07	Left SHift • AQ...........	Shift (AQ) Left by Y
10	ENTer • Q	Y → Q
11	ENTer • A................	Y → A
12	ENTer • B^n	Y → B^j
13^	EXternal - FunCTion • C^n.......	ĵ=0 or 1, (Y) → C$^{\hat{j}}$, ĵ=0 or 1, See Note
14	STore • Q	(Q) → Y; k=0, Q' → Q
15	STore • A	(A) → Y; k=4, A' → A
16	STore • B^n	(B)^j → Y
17^	STore • C^n...............	(C)^j → Y
20	ADD • A................	(A) + Y → A
21	SUBtract • A...............	(A) − Y → A
22	MULtiply	(Q) Y → AQ
23*	DIVide	(AQ) / Y → Q, R → A$_f$
24	RePLace • A + Y..........	(A) + (Y) → Y & A
25	RePLace • A − Y..........	(A) − (Y) → Y & A
26*	ADD • Q	(Q) + Y → Q, (A)$_j$ = (A)$_f$ } j interpretation
27*	SUBtract • Q	(Q) − Y → Q, (A)$_j$ = (A)$_f$ } reversed for ABQ
30	ENTer • Y + Q............	Y + (Q) → A
31	ENTer • Y − Q............	Y − (Q) → A
32	STore • A + Q............	(A) + (Q) → Y & A
33	STore • A − Q............	(A) − (Q) → Y & A
34	RePLace • Y + Q............	(Y) + (Q) → Y & A
35	RePLace • Y − Q............	(Y) − (Q) → Y & A
36	RePLace • Y + 1............	(Y) + 1 → Y & A
37	RePLace • Y − 1............	(Y) − 1 → Y & A
40*	ENTer • LP**	L[Y(Q)] → A; j=2, even parity; j=3, odd parity
41	ADD • LP	L[Y(Q)] + (A) → A
42	SUBtract • LP...........	(A) − L[Y(Q)] → A
43	COMpare • MASK...........	(A) − L[Y(Q)] SENSE (j) L(A) + L[Y(Q)]; (A)$_j$ = (A)$_f$
44*	RePLace • LP...........	L(Y)(Q) → Y & A; j=2, even parity; j=3, odd parity
45	RePLace • A + LP...........	L(Y)(Q) + (A) → Y & A
46	RePLace • A − LP...........	(A) − L(Y)(Q) → Y & A
47	STore • LP...........	L(A)(Q) → Y; (A)$_j$ = (A)$_f$
50	SELective • SET	SET (A)$_n$ FOR Y$_n$ = 1
51	SELective • CP**	COMPLEMENT (A)$_n$ FOR Y$_n$ = 1
52	SELective • CL**	CLEAR (A)$_n$ FOR Y$_n$ = 1
53	SELective • SU**	Y$_n$ → (A)$_n$ FOR (Q)$_n$ = 1
54	Replace SElective • SET.....	SET (A)$_n$ FOR (Y)$_n$ = 1, → Y & A
55	Replace SElective • CP.....	COMPLEMENT (A)$_n$ FOR (Y)$_n$ = 1, → Y & A
56	Replace SElective • CL.....	CLEAR (A)$_n$ FOR (Y)$_n$ = 1, → Y & A
57	Replace SElective • SU.....	(Y)$_n$ → (A)$_n$ FOR (Q)$_n$ = 1, → Y
60	JumP (arithmetic)	Jump to Y if j-condition is satisfied.
61	JumP (manual)...........	(see JP & RJP j - Designators)
62^	JumP (if • C^n has ACTIVE INput buffer)	Jump to Y if C$^{\hat{j}}$ input buffer active (see JP & RJP j - Designators)
63^	JumP (if • C^n has ACTIVE OUTput buffer)	Jump to Y if C$^{\hat{j}}$ output buffer active (see JP & RJP j - Designators)
64	Return JumP (arithmetic)	Jump to Y + 1 and P + 1 → Y$_L$ if j condition is
65	Return JumP (manual).....	satisfied (see JP & RJP j - Designators)
66^	TERMinate • C^n • INPUT.....	Terminate input buffer on channel ĵ
67^	TERMinate • C^n • OUTPUT	Terminate output buffer on channel ĵ
70*	RePeaT.................	Execute NI Y times
71	BSKip • B^n................	(B)^j = Y, skip NI and clear (B)^j; (B)^j ≠ Y, Advance B^j and read NI
72	BJumP • B^n................	(B)^j = 0, read NI; (B)^j ≠ 0, (B)^j − 1 and jump to address Y
73^	INput • C^n (without monitor mode).	Buffer IN on C$^{\hat{j}}$; k̂ = 3, (Y) → (00100 + ĵ); k̂ = 1, (Y)$_L$ → (00100 + ĵ); k̂ = 0, Y → (00100 + ĵ)$_L$.
74^	OUTput • C^n (without monitor mode).	Buffer OUT on C$^{\hat{j}}$; k̂ = 3, (Y) → (00120 + ĵ); k̂ = 1, (Y)$_L$ → (00120 + ĵ)$_L$; k̂ = 0, Y → (00120 + ĵ)$_L$.
75^	INput • C^n (with • MONITOR mode).	Buffer IN on C$^{\hat{j}}$ with mon. k̂ = 3, (Y) → (00100 + ĵ); k̂ = 1, (Y)$_L$ → (00100 + ĵ)$_L$; k̂ = 0, Y → (00100 + ĵ)$_L$. mon. inter. at 00040 + j
76^	OUTput • C^n (with • MONITOR mode).	Buffer OUT on C$^{\hat{j}}$ with mon. k̂ = 3, (Y) → (00120 + ĵ); k̂ = 1, (Y)$_L$ → (00120 + ĵ)$_L$; k̂ = 0, Y → (00120 + ĵ)$_L$. mon. inter. at 00060 + ĵ
—	NO - OPeration	CS-1 Mono-codes
—	ComPlement • A or • Q.......	CS-1 Mono-codes
—	CLear • A, • Q, • B^n, or Y.....	CS-1 Mono-codes

**LP - Logical Product CP - Complement SU - Substitute CL - Clear * ^ } Special j & k Designators (see opposite side of card) Y - The operand; Y or (Y)

NOTE: Skip NI if other Computer (on channel 0 or 1) has input buffer active. Execute twice.

NTDS UNIT COMPUTER

AN/USQ-20

Repertoire of Instructions

01	Right SHift • Q	Shift (Q) Right by Y
02	Right SHift • A	Shift (A) Right by Y
03	Right SHift • AQ	Shift (AQ) Right by Y
04*	COMpare • A, • Q, • AQ	Sense (j); $(A)_j = (A)_f$
05	Left SHift • Q	Shift (Q) Left by Y
06	Left SHift • A	Shift (A) Left by Y
07	Left SHift • AQ	Shift (AQ) Left by Y
10	ENTer • Q	Y → Q
11	ENTer • A	Y → A
12	ENTer • B^n	Y → B^j
13^	EXternal - FunCTion • C^n	$\hat{j} \neq 0$ or 1, (Y) → $C^{\hat{j}}$, $\hat{j}=0$ or 1, See Note.
14	STo Re • Q	(Q) → Y; k=0, Q' → Q
15	STo Re • A	(A) → Y; k=4, A' → A
16	STo Re • B^n	$(B)^j$ → Y
17^	STo Re • C^n	$(C)^{\hat{j}}$ → Y
20	ADD • A	(A) + Y → A
21	SUBtract • A	(A) − Y → A
22	MULtiply	(Q) Y → AQ
23*	DIVide	(AQ) / Y → Q. R → A_f
24	RePLace • A+Y	(A) + (Y) → Y & A
25	RePLace • A−Y	(A) − (Y) → Y & A
26*	ADD • Q	(Q) + Y → Q. $(A)_j = (A)_f$ } j interpretation
27*	SUBtract • Q	(Q) − Y → Q. $(A)_j = (A)_f$ } reversed for A&Q
30	ENTer • Y+Q	Y + (Q) → A
31	ENTer • Y−Q	Y − (Q) → A
32	STo Re • A+Q	(A) + (Q) → Y & A
33	STo Re • A−Q	(A) − (Q) → Y & A
34	RePLace • Y+Q	(Y) + (Q) → Y & A
35	RePLace • Y−Q	(Y) − (Q) → Y & A
36	RePLace • Y+1	(Y) + 1 → Y & A
37	RePLace • Y−1	(Y) − 1 → Y & A
40*	ENTer • LP**	L[Y(Q)] → A; j=2, even parity, j=3, odd parity
41	ADD • LP	L[Y(Q)] + (A) → A
42	SUBtract • LP	(A) − L[Y(Q)] → A
43	COMpare • MASK	(A) − L[Y(Q)] SENSE (j), (A) + L[Y(Q)]; $(A)_j = (A)_f$
44*	RePLace • LP	L(Y)(Q) → Y & A; j=2, even parity; j=3, odd parity
45	RePLace • A+LP	L(Y)(Q) + (A) → Y & A
46	RePLace • A−LP	(A) − L(Y)(Q) → Y & A
47	STo Re • LP	L(A)(Q) → Y; $(A)_j = (A)_f$
50	SELective • SET	SET $(A)_n$ FOR $Y_n = 1$
51	SELective • CP**	COMPLEMENT $(A)_n$ FOR $Y_n = 1$
52	SELective • CL**	CLEAR $(A)_n$ FOR $Y_n = 1$
53	SELective • SU**	Y_n → $(A)_n$ FOR $(Q)_n = 1$
54	Replace SElective • SET	SET $(A)_n$ FOR $(Y)_n = 1$, → Y & A
55	Replace SElective • CP	COMPLEMENT $(A)_n$ FOR $(Y)_n = 1$, → Y & A
56	Replace SElective • CL	CLEAR $(A)_n$ FOR $(Y)_n = 1$, → Y & A
57	Replace SElective • SU	$(Y)_n$ → $(A)_n$ FOR $(Q)_n = 1$, → Y
60	JumP (arithmetic)	} Jump to Y if j-condition is satisfied.
61	JumP (manual)	} (see JP & RJP j - Designators)
62^	JumP (if • C^n has ACTIVE INput buffer)	Jump to Y if $C^{\hat{j}}$ input buffer active } (see JP & RJP j - Designators)
63^	JumP (if • C^n has ACTIVE OUTput buffer)	Jump to Y if $C^{\hat{j}}$ output buffer active } (see JP & RJP j - Designators)
64	Return JumP (arithmetic)	} Jump to Y+1 and P+1 → Y_L if j condition is
65	Return JumP (manual)	} satisfied (see JP & RJP j - Designators)
66^	TERMinate • C^n • INPUT	Terminate input buffer on channel $\hat{j}$
67^	TERMinate • C^n • OUTPUT	Terminate output buffer on channel $\hat{j}$
70*	RePeaT	Execute NI Y times
71	BSKip • B^n	$(B)^j$ = Y, skip NI and clear $(B)^j$; $(B)^j \neq$ Y, Advance B^j and read NI
72	BJumP • B^n	$(B)^j = 0$, read NI; $(B)^j \neq 0$, $(B)^j - 1$ and jump to address Y
73^	INput • C^n (without monitor mode)	Buffer IN on $C^{\hat{j}}$; $\hat{k}=3$, (Y) → (00100 + $\hat{j}$); $\hat{k}=1$, $(Y)_L$ → (00100 + $\hat{j}$); $\hat{k}=0$, Y → (00100 + $\hat{j}$)$_L$.
74^	OUTput • C^n (without monitor mode)	Buffer OUT on $C^{\hat{j}}$; $\hat{k}=3$, (Y) → (00120 + $\hat{j}$); $\hat{k}=1$, $(Y)_L$ → (00120 + $\hat{j}$)$_L$; $\hat{k}=0$, Y → (00120 + $\hat{j}$)$_L$.
75^	INput • C^n (with • MONITOR mode)	Buffer IN on $C^{\hat{j}}$ with mon. $\hat{k}=3$, (Y) → (00100 + $\hat{j}$); $\hat{k}=1$, $(Y)_L$ → (00100 + $\hat{j}$)$_L$; $\hat{k}=0$, Y → (00100 + $\hat{j}$)$_L$. mon. inter. at 00040 + $\hat{j}$
76^	OUTput • C^n (with • MONITOR mode)	Buffer OUT on $C^{\hat{j}}$ with mon. $\hat{k}=3$, (Y) → (00120 + $\hat{j}$); $\hat{k}=1$, $(Y)_L$ → (00120 + $\hat{j}$)$_L$; $\hat{k}=0$, Y → (00120 + $\hat{j}$)$_L$. mon. inter. at 00060 + $\hat{j}$
	— NO − OPeration	} CS-1 Mono − codes
	— ComPlement • A or • Q	} CS-1 Mono − codes
	— CLear • A, • Q, • B^n, or Y	} CS-1 Mono − codes

**LP - Logical Product CP - Complement SU - Substitute CL - Clear *^ } Special j & k Designators (see opposite side of card) Y - The operand; Y or (Y)

NOTE: Skip NI if other Computer (on channel 0 or 1) has input buffer active. Execute twice.

NTDS UNIT COMPUTER

AN/USQ-20 *Repertoire of Instructions*

JP & RJP j-DESIGNATORS

j	JP 60 RJP 64	JP 61 RJP 65
0	(No Jump)*	(Uncond. Jump)
1	(Uncond. Jump)*	KEY 1
2	Q POS	KEY 2
3	Q NEG	KEY 3
4	A ZERO	STOP
5	A NOT Zero	STOP 5
6	A POS	STOP 6
7	A NEG	STOP 7
$\hat{j}$	62 $\hat{j}$	63 $\hat{j}$
$0\text{-}15_8$	C^n ACTIVE IN	C^n ACTIVE OUT

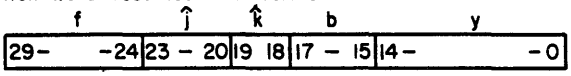
*60 Clears interrupt & bootstrap modes.

$\hat{j}$-DESIGNATORS

(4 bits)

$\hat{j}$ Occupies 4 bit positions and represents C^n where n may be $0—15_8$

The instruction word assumes the format:



$\hat{k}$-DESIGNATORS

(2 bits)

$\hat{k}$	EX—FCT 13	STR•C^n 17	JP 62 63	IN•C^n, OUT•C^n 73 75 74 76
0	'not used'	'not used'	'blank'	'blank'
1	'not used'	'not used'	L	L
2	'not used'	'not used'	U	'not used'
3	W	W	W	W

* j-DESIGNATORS

j	COM•A, •Q, •AQ 04	DIV 23	ADD•Q, SUB•Q 26 27	ENT•LP, RPL•LP 40 44	RPT 70	
0	(no skip)	(no skip)	(no skip)	(no skip)	(no mod.)	Y of NE = Y
1	(unconditional skip)	SKIP	SKIP	SKIP	ADV	Y of NE = Y+1
2	Y LESS ¦ Y ≤ (Q)	NO Over Flow	A POS	EVEN parity	BACK	Y of NE = Y−1
3	Y MORE ¦ Y > (Q)	Over Flow	A NEG	ODD parity	ADD B	Y of NE = Y+B^b
4	Y IN ¦ (Q) ≥ Y and Y > (A)	A ZERO	Q ZERO	A ZERO	Rpl. Inc.	Y of NE = Y [+B^6] ✓
5	Y OUT ¦ (Q) < Y or Y ≤ (A)	A NOT Zero	Q NOT Zero	A NOT Zero	ADV R	Y of NE = Y+1 [+B^6] ✓
6	Y LESS ¦ Y ≤ (A)	A POS	Q POS	A POS	BACK R	Y of NE = Y−1 [+B^6] ✓
7	Y MORE ¦ Y > (A)	A NEG	Q NEG	A NEG	ADD B R	Y of NE = Y+B^b [+B^6] ✓

✓ B^6 Increment if NI is RPL class; increments Y address for the store portion of the replace.

NE — Next execution

NORMAL j-DESIG.

j	(Not applicable on * or ^) Skip Code
0	(no skip)
1	SKIP
2	Q POS
3	Q NEG
4	A ZERO
5	A NOT Zero
6	A POS
7	A NEG

NORMAL k-DESIGNATORS

k	READ Code	READ Origin	STORE Code	STORE Dest.	REPLACE Code	REPLACE Origin	REPLACE Dest.
0	'blank'	U_L	Q	Q	'not used'	—	—
1	L	M_L	L	M_L	L	M_L	M_L
2	U	M_U	U	M_U	U	M_U	M_U
3	W	M	W	M	W	M	M
4	X	XU_L	A	A	'not used'	—	—
5	LX	XM_L	CPL	Cpl M_L	LX	XM_L	M_L
6	UX	XM_U	CPU	Cpl M_U	UX	XM_U	M_U
7	A	A	CPW	Cpl M	'not used'	—	—

LEGEND

M — Memory word (30 bits)

M_L — Lower half memory word

M_U — Upper half memory word

X — Sign bit extended

Cpl — Complement

A — A-register

Q — Q-register

U — U-register