

APOLLO GUIDANCE COMPUTER

Information Series

ISSUE 32

BLOCK II MACHINE INSTRUCTIONS

FR-2-132

29 October 1965

CONTENTS

Paragraph		Page
32-1	INTRODUCTION	32-1
32-3	EXECUTION OF INSTRUCTIONS.	32-19
32-5	Execution of Subinstructions.	32-19
32-9	Control Pulses	32-20
32-16	Subinstruction STD2	32-20
32-21	Data Transfer Diagrams	32-21
32-25	Example of Instruction Executions	32-23
32-30	REGULAR INSTRUCTIONS.	32-53
32-31	Sequence Changing Instructions.	32-53
32-32	Instruction TC K	32-53
32-38	Instruction TCF F	32-54
32-44	Instruction CCS E	32-56
32-50	Instruction BZF F	32-62
32-56	Instruction BZMF F	32-66
32-61	Fetching and Storing Instructions	32-67
32-62	Instruction CA K	32-67
32-68	Instruction CS K	32-69
32-73	Instruction DCA K	32-72
32-79	Instruction DCS K.	32-74
32-84	Instruction TS E	32-75
32-92	Instruction XCH E	32-79
32-98	Instruction LXCH E	32-81
32-103	Instruction QXCH E	32-82
32-108	Instruction DXCH E	32-83
32-114	Modifying Instructions	32-87
32-115	Instruction NDX E	32-87
32-122	Instruction NDX K	32-88
32-127	Arithmetic and Logic Instructions.	32-92
32-128	Instruction AD K	32-92
32-134	Instruction SU E	32-94
32-139	Instruction MP K	32-95
32-143	Principle of Operation	32-96
32-148	Actual Execution	32-97
32-154	Instruction DV E	32-112

CONTENTS (cont)

Paragraph		Page
32-158	Principle of Operation.	32-113
32-162	Actual Execution	32-114
32-171	Instruction ADS E.	32-132
32-177	Instruction DAS E.	32-134
32-186	Instruction INCR E	32-138
32-192	Instruction AUG E	32-141
32-197	Instruction DIM E	32-143
32-202	Instruction MSU E	32-144
32-212	Instruction MSK K	32-149
32-218	Channel Instructions	32-151
32-219	Instruction READ H	32-151
32-225	Instruction WRITE H.	32-153
32-230	Instruction RAND H	32-154
32-236	Instruction WAND H	32-157
32-241	Instruction ROR H	32-158
32-247	Instruction WOR H	32-160
32-252	Instruction RXOR H	32-161
32-258	Special Instructions	32-163
32-259	Instruction EXTEND.	32-163
32-263	Instruction INHINT	32-165
32-267	Instruction RELINT	32-166
32-271	Instruction RESUME.	32-166
32-275	Instructions CYR, SR, CYL, and EDOP	32-169
32-277	INVOLUNTARY INSTRUCTIONS	32-171
32-278	Interrupting Instructions	32-171
32-279	Instruction RUPT.	32-171
32-283	Instruction GO.	32-174
32-286	Counter Instructions	32-174
32-287	Instruction PINC C	32-174
32-293	Instruction MINC C	32-176
32-297	Instruction DINC C	32-176
32-301	Instruction PCDU C	32-177
32-305	Instruction MCDU C	32-178
32-309	Instruction SHINC C	32-179
32-315	Instruction SHANC C.	32-181
32-319	PERIPHERAL INSTRUCTIONS	32-183

CONTENTS (cont)

Paragraph		Page
32-320	Sequence Changing Test Instructions	32-183
32-321	Instruction TCSAJ K	32-183
32-325	Display and Load Test Instructions	32-183
32-326	Instruction FETCH K	32-183
32-329	Instruction STORE E	32-184
32-332	Instruction INOTRD H	32-187
32-335	Instruction INOTLD H	32-187

ILLUSTRATIONS

Figure		Page
32-1	Subinstruction STD2	32-22
32-2	Subinstruction TC0	32-55
32-3	Subinstruction CCS0, Branch on Quantity Greater than Plus Zero	32-58
32-4	Subinstruction CCS0, Branch on Plus Zero	32-59
32-5	Subinstruction CCS0, Branch on Quantity Less than Minus Zero.	32-60
32-6	Subinstruction CCS0, Branch on Minus Zero.	32-61
32-7	Subinstruction BZF0, With Register A Containing a Positive Non-Zero Quantity	32-64
32-8	Subinstruction BZF0, With Register A Containing Plus Zero.	32-65
32-9	Subinstruction CA0	32-68
32-10	Subinstruction DCA0	32-72
32-11	Subinstruction DCA1	32-73
32-12	Subinstruction TS0, Without Overflow Bit in Register A	32-77
32-13	Subinstruction TS0, With Positive Overflow Bit in Register A.	32-78
32-14	Subinstruction XCH0	32-80
32-15	Subinstruction DXCH0	32-85
32-16	Subinstruction DXCH1	32-86
32-17	Subinstruction NDX0	32-89
32-18	Subinstruction NDX1	32-90
32-19	Subinstruction AD0	32-93
32-20	Positive Product, Principle of Multiplication	32-98

ILLUSTRATIONS (cont)

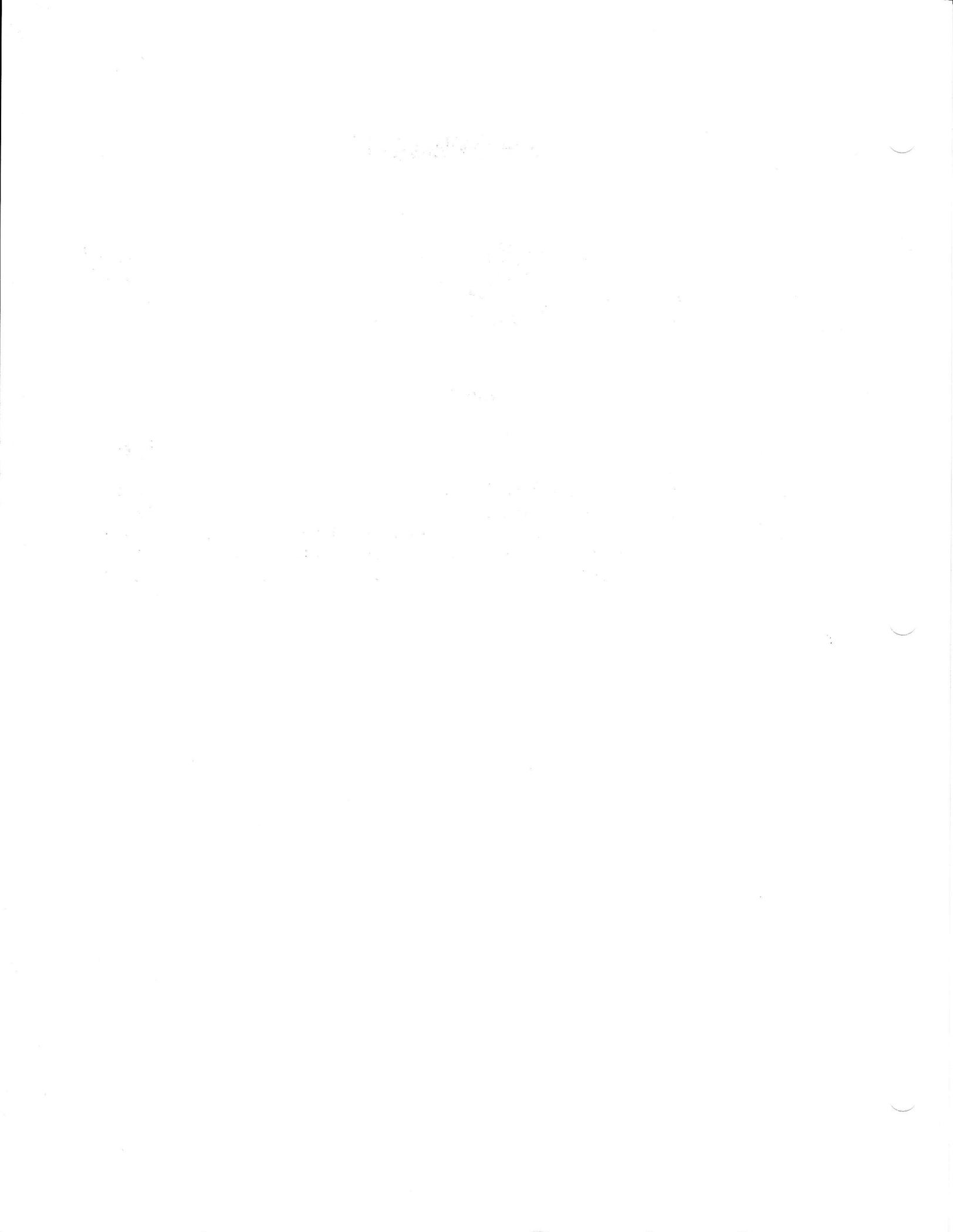
Figure		Page
32-21	Negative Product, Principle of Multiplication	32-99
32-22	Subinstruction MP0, With Two Positive Quantities	32-100
32-23	Subinstruction MP0, With Positive Quantity in A and Negative Quantity in E	32-101
32-24	Subinstruction MP0, With Negative Quantity in A and Positive Quantity in E	32-102
32-25	Subinstruction MP0, With Two Negative Quantities.	32-103
32-26	Subinstruction MP1, Positive Product	32-104
32-27	Subinstruction MP3, Positive Product	32-105
32-28	Subinstruction MP1, Negative Product.	32-106
32-29	Subinstruction MP3, Negative Product.	32-107
32-30	Positive Product, Actual Multiplication	32-108
32-31	Negative Product, Actual Multiplication	32-110
32-32	Principle of Division, Manual Method	32-116
32-33	Principle of Division, Machine Method.	32-117
32-34	Divide Instruction, Flow Diagram	32-119
32-35	Subinstruction DV0	32-123
32-36	Subinstruction DV1	32-124
32-37	Subinstruction DV3	32-125
32-38	Subinstruction DV7	32-126
32-39	Subinstruction DV6	32-127
32-40	Subinstruction DV4	32-128
32-41	Actual Division	32-129
32-42	Subinstruction ADS0	32-133
32-43	Subinstruction DAS0	32-136
32-44	Subinstruction DAS1	32-137
32-45	Subinstruction INCR0	32-140
32-46	Subinstruction MSU0	32-146
32-47	Subinstruction MSK0	32-150
32-48	Subinstruction READ0	32-152
32-49	Subinstruction RAND0	32-156
32-50	Subinstruction ROR0	32-159
32-51	Subinstruction RXOR0	32-162
32-52	Subinstruction STD2, Preceding Instruction EXTEND.	32-164
32-53	Subinstruction NDX0 of Instruction RESUME.	32-167
32-54	Subinstruction RSM3	32-168
32-55	Subinstruction RUPT0	32-172
32-56	Subinstruction RUPT1	32-173

ILLUSTRATIONS (cont)

Figure		Page
32-57	Subinstruction PINC	32-175
32-58	Subinstruction SHINC	32-180
32-59	Subinstruction FETCH0	32-185
32-60	Subinstruction FETCH1	32-186

TABLES

Table		Page
32-1	Machine Instruction Types	32-1
32-2	Machine Instructions.	32-2
32-3	Machine Instructions, Alphabetical Listing	32-13
32-4	Control Pulses Generated at Various Actions	32-25
32-5	Control Pulses.	32-47



32-1. INTRODUCTION

32-2. This is the thirty-second issue of the AGCIS published to inform the technical staff at MIT/IL and Raytheon about the Apollo guidance computer (AGC) subsystems. The various Block II instruction types and the order code structure of Machine Instructions are discussed in paragraphs 30-148 through 30-187 of Issue 30. Issue 32 analyzes the operations performed by the Machine Instructions. Table 32-1 (shown below) briefly reviews Machine Instruction types. Table 32-2 contains a functional description of all Machine Instructions and table 32-3 lists all Machine Instructions alphabetically for quick reference.

Table 32-1
MACHINE INSTRUCTION TYPES

Group	Type	
Regular Instructions	Basic Instructions Extra Code Instructions	Sequence changing instructions Fetching and storing instructions Modifying instructions Arithmetic and logic instructions
	Channel Instructions Special Instructions	
Involuntary Instructions	Interrupting Instructions Counter Instructions	
Peripheral Instructions	Sequence changing test instructions Display and load test instructions	

TABLE 32-2
MACHINE INSTRUCTIONS

① Symbolic Instruction Word	② Order Code	③ Sub- instructions Executed	Description ① ④										
REGULAR INSTRUCTIONS													
Sequence Changing Instructions													
TC K	00.	TC0	"Transfer Control to K" Takes next instruction from K and stores return address (I+1) in Q.										
TCF F	01.2 01.4 01.6	TCF0	"Transfer Control to Fixed F" Takes next instruction from F without changing c(Q).										
CCS E	01.0	CCS0 STD2	"Count, Compare, and Skip on E" Branches according to c(E) and stores in A the c(E) diminished by one. <table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center; border-bottom: 1px solid black;">c(E)</th> <th style="text-align: center; border-bottom: 1px solid black;">Transfers to</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">positive nonzero</td> <td style="text-align: center;">I+1</td> </tr> <tr> <td style="text-align: center;">plus zero</td> <td style="text-align: center;">I+2</td> </tr> <tr> <td style="text-align: center;">negative nonzero</td> <td style="text-align: center;">I+3</td> </tr> <tr> <td style="text-align: center;">minus zero</td> <td style="text-align: center;">I+4</td> </tr> </tbody> </table>	c(E)	Transfers to	positive nonzero	I+1	plus zero	I+2	negative nonzero	I+3	minus zero	I+4
c(E)	Transfers to												
positive nonzero	I+1												
plus zero	I+2												
negative nonzero	I+3												
minus zero	I+4												
BZF F	16.2 16.4 16.6	BZF0 STD2	"Branch on Zero to Fixed F" Branches according to c(A). <table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center; border-bottom: 1px solid black;">c(A)</th> <th style="text-align: center; border-bottom: 1px solid black;">Transfers to</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">plus or minus zero</td> <td style="text-align: center;">F (subinstruction STD2 is not executed)</td> </tr> <tr> <td style="text-align: center;">non zero</td> <td style="text-align: center;">I+1 (subinstruction STD2 is executed)</td> </tr> </tbody> </table>	c(A)	Transfers to	plus or minus zero	F (subinstruction STD2 is not executed)	non zero	I+1 (subinstruction STD2 is executed)				
c(A)	Transfers to												
plus or minus zero	F (subinstruction STD2 is not executed)												
non zero	I+1 (subinstruction STD2 is executed)												

TABLE 32-2
MACHINE INSTRUCTIONS (cont)

① Symbolic Instruction Word	② Order Code	③ Sub- instructions Executed	④ Description						
REGULAR INSTRUCTIONS									
Sequence Changing Instructions (cont)									
BZMF F	12.2 12.4 12.6	BZMF0 STD2	<p>"Branch on Zero or Minus to Fixed F" Branches according to c(A).</p> <table border="1" data-bbox="917 829 1437 1108"> <thead> <tr> <th data-bbox="917 829 1161 892">c(A)</th> <th data-bbox="1161 829 1437 892">Transfers to</th> </tr> </thead> <tbody> <tr> <td data-bbox="917 892 1161 997">zero or negative nonzero</td> <td data-bbox="1161 892 1437 997">F (subinstruction STD2 is not executed)</td> </tr> <tr> <td data-bbox="917 997 1161 1108">positive nonzero</td> <td data-bbox="1161 997 1437 1108">I+1 (subinstruction STD2 is executed)</td> </tr> </tbody> </table>	c(A)	Transfers to	zero or negative nonzero	F (subinstruction STD2 is not executed)	positive nonzero	I+1 (subinstruction STD2 is executed)
c(A)	Transfers to								
zero or negative nonzero	F (subinstruction STD2 is not executed)								
positive nonzero	I+1 (subinstruction STD2 is executed)								
Fetching and Storing Instructions									
CA K	03.	CA0 STD2	"Clear and Add K" Enters c(K) into A. Takes next instruction from I+1.						
CS K	04.	CS0 STD2	"Clear and Subtract K" Enters the complemented c(K) into A. Takes next instruction from I+1.						
DCA K	13.	DCA0 DCA1 STD2	"Double Clear and Add K" Enters c(K, K+1) into A and L. Takes next instruction from I+1.						
DCS K	14.	DCS0 DCS1 STD2	"Double Clear and Subtract K" Enters the complemented c(K, K+1) into A and L. Takes next instruction from I+1.						

TABLE 32-2
MACHINE INSTRUCTIONS (cont)

$\triangle 1$ Symbolic Instruction Word	$\triangle 2$ Order Code	$\triangle 3$ Sub- instructions Executed	Description $\triangle 1$ $\triangle 4$
REGULAR INSTRUCTIONS			
Fetching and Storing Instructions (cont)			
TS E	05.4	TS0 STD2	"Transfer to Storage E" If A does not contain an overflow quantity, instruction enters c(A) into E and takes next instruction from I+1. If A contains a positive overflow, instruction enters c(A) without overflow bit into E, enters plus one into A, and takes next instruction from I+2. If A contains a negative overflow, instruction enters c(A) without overflow bit into E, enters minus one into A, and takes next instruction from I+2.
XCH E	05.6	XCH0 STD2	"Exchange A and E" Exchanges c(A) with c(E). Takes next instruction from I+1.
LXCH E	02.2	LXCH0 STD2	"Exchange L and E" Exchanges c(L) with c(E). Takes next instruction from I+1.
QXCH E	12.2	QXCH0 STD2	"Exchange Q and E" Exchanges c(Q) with c(E). Takes next instruction from I+1.
DXCH E	05.2	DXCH0 DXCH1 STD2	"Double Exchange A and E" Exchanges c(A, L) with c(E, E+1). Takes next instruction from I+1.

TABLE 32-2
MACHINE INSTRUCTIONS (cont)

① Symbolic Instruction Word	② Order Code	③ Sub- instructions Executed	① Description ④
REGULAR INSTRUCTIONS			
Modifying Instructions			
NDX E	05.0	NDX0 NDX1	"Index Next Basic Instruction with E" Adds c(E) to c(I+1) and takes sum as next instruction.
NDX K	15.	NDXX0 NDXX1	"Index Next Extra-Code Instruction with K" Adds c(K) to c(I+2) and takes sum as next instruction. Retains the ONE in bit position SQ-EXT.
Arithmetic and Logic Instructions			
AD K	06.	AD0 STD2	"Add K" Adds c(K) to c(A) and stores sum in A. Takes next instruction from I+1.
SU E	16.0	SU0 STD2	"Subtract E" Subtracts c(E) from c(A) and stores the difference in A. Takes next instruction from I+1.
MP K	17.	MP0 MP1 MP3	"Multiply K" Multiplies c(K) by c(A) and stores double precision product in A and L (signs in A and L agree). Takes next instruction from I+1.

TABLE 32-2
MACHINE INSTRUCTIONS (cont)

$\triangle 1$ Symbolic Instruction Word	$\triangle 2$ Order Code	$\triangle 3$ Sub- instructions Executed	Description $\triangle 1$ $\triangle 4$
REGULAR INSTRUCTIONS			
Arithmetic and Logic Instructions (cont)			
DV E	11.0	DV0 } DV1 } DV3 } DV7 } $\triangle 3$ DV6 } DV4 } $\triangle 5$ STD2 }	"Divide by E" Divides double precision quantity $c(A, L)$ by $c(E)$, stores quotient in A and remainder in L. Takes next instruction from I+1. Signs of $b(A)$ and $b(L)$ need not agree. Sign of remainder equals sign of dividend.
ADS E	02.6	ADS0 STD2	"Add to Storage E" Adds $c(A)$ and $c(E)$, stores sum with overflow bit in A and sum without overflow bit in E.
DAS E	02.0	DAS0 DAS1 STD2	"Double Add to Storage E" Adds $c(A, L)$ and $c(E, E+1)$ and stores sum without overflow bit in E and E+1. Enters plus one into A in case of positive overflow, minus one in case of negative overflow, and plus zero in case of no overflow. Enters plus zero into L and takes next instruction from I+1.
INCR E	02.4	INCR0 STD2	"Increment E" Adds plus one to $c(E)$ and stores incremented quantity in E. Takes next instruction from I+1.
AUG E	12.4	AUG0 STD2	"Augment E" Increases the magnitude of the quantity contained in E by one and stores the augmented quantity in E. Takes next instruction from I+1.

TABLE 32-2
MACHINE INSTRUCTIONS (cont)

① Symbolic Instruction Word	② Order Code	③ Sub- instructions Executed	Description ① ④
REGULAR INSTRUCTIONS			
Arithmetic and Logic Instructions (cont)			
DIM E	12.6	DIM0 STD2	"Diminish E" Decreases the magnitude of the quantity contained in E by one and stores diminished quantity in E. Takes next instruction from I+1.
MSU E	12.0	MSU0 STD2	"Modular Subtract E" Subtracts cyclic TWO's complement number in E from cyclic TWO's complement number in A and stores difference expressed in ONE's complement number in A. Takes next instruction from I+1.
MSK K	07.	MSK0 STD2	"Mask with K" AND's c(A) with c(K) and stores logical product in A. Takes next instruction from I+1.
Channel Instructions			
READ H	10.0	READ0 STD2	"Read H" Enters c(H) into A. Takes next instruction from I+1.
WRITE H	10.1	WRITE0 STD2	"Write H" Enters c(A) into H. Takes next instruction from I+1.
RAND H	10.2	RAND0 STD2	"Read and AND H" AND's c(A) and c(H) and stores logical product in A. Takes next instruction from I+1.

TABLE 32-2
MACHINE INSTRUCTIONS (cont)

① Symbolic Instruction Word	② Order Code	③ Sub- instructions Executed	④ Description
REGULAR INSTRUCTIONS			
Channel Instructions (cont)			
WAND H	10.3	WAND0 STD2	"Write and AND H" AND's c(A) and c(H), and stores logical product in A and H. Takes next instruction from I+1.
ROR H	10.4	ROR0 STD2	"Read and OR H" OR's c(A) and c(H), and stores logical sum in A. Takes next instruction from I+1.
WOR H	10.5	WOR0 STD2	"Write and OR H" OR's c(A) and c(H), and stores logical sum in A and H. Takes next instruction from I+1.
RXOR H	10.6	RXOR0 STD2	"Read and Exclusive OR H" Forms exclusive OR from c(A) and c(H), and stores result in A. Takes next instruction from I+1.
Special Instructions			
EXTEND	00.0006	STD2	"Extend" Enters a ONE into bit position SQ-EXT. The next instruction, taken from I+1, is an Extra-Code Instruction.
INHINT	00.0004	STD2	"Inhibit Interrupt" Sets inhibit interrupt switch in Interrupt Priority Control to prevent interruption of program execution. Takes next instruction from I+1.

TABLE 32-2
MACHINE INSTRUCTIONS (cont)

① Symbolic Instruction Word	② Order Code	③ Sub- instructions Executed	① Description ④
REGULAR INSTRUCTIONS			
Special Instructions (cont)			
RELINT	00.0003	STD2	"Release Inhibit Interrupt" Resets inhibit interrupt switch to allow program interruption in favor of a programmed operation of higher priority. Takes next instruction from I+1.
RESUME	05.0017	NDX0 RSM3	"Resume Interrupted Program" Takes next instruction from location 0017 and enters content of location 0015 into Z. Thus, execution of the interrupted program section is resumed.
CYR	.0020	⑥	"Cycle Right" Cycles quantity, which is entered into location 0020, one place to the right.
SR	.0021	⑥	"Shift Right" Shifts quantity, which is entered into location 0021, one place to the right.
CYL	.0022	⑥	"Cycle Left" Cycles quantity, which is entered into location 0022, one place to the left.
EDOP	.0023	⑥	"Edit Operator" Shifts quantity, which is entered into location 0023, seven places to the right.

TABLE 32-2
MACHINE INSTRUCTIONS (cont)

① Symbolic Instruction Word	② Order Code	③ Sub- instructions Executed	④ Description
INVOLUNTARY INSTRUCTIONS			
Interrupting Instructions			
RUPT	10.7	RUPT0 RUPT1 STD2	"Interrupt Program Execution" Takes next instruction from address supplied by Interrupt Priority Control. Stores c(B) in location 0017 and c(Z) in location 0015.
GO	00. 00.4000	GOJ1 TC0	"Go" Takes next instruction from loca- tion 04000 in E Memory.
Counter Instructions			
PINC C	none	PINC	"Plus Increment C" Adds one to c(C) and stores incremented quantity in C.
MINC C	none	MINC	"Minus Increment C" Subtracts one from c(C) and stores decremented quantity in C.
DINC C	none	DINC	"Diminish Increment C" De- creases the magnitude of the quantity contained in C by one and stores diminished quantity in C.
PCDU C	none	PCDU	"Plus CDU C" Adds one to cyclic TWO's com- plement number in C and stores incremented quantity in C.

TABLE 32-2
MACHINE INSTRUCTIONS (cont)

① Symbolic Instruction Word	② Order Code	③ Sub- instructions Executed	④ Description
INVOLUNTARY INSTRUCTIONS			
Counter Instructions (cont)			
MCDU C	none	MCDU	"Minus CDU C" Subtracts one from cyclic TWO's complement number in C and stores decremented quantity in C.
SHINC C	none	SHINC	"Shift Increment C" Shifts c(C) one place to the left and enters a ZERO into bit position 0 of C.
SHANC C	none	SHANC	"Shift and Add Increment C" Shifts c(C) one place to the left and enters a ONE into bit position 0 of C.
PERIPHERAL INSTRUCTIONS			
TCSAJ K	00.	TCSAJ3 STD2	"Transfer control to specified address K" Takes next instruction from address which is supplied by GSE.
FETCH K	none	FETCH0 FETCH1	"Fetch K"; displays c(K) on GSE. Address K is supplied by GSE.
STORE E	none	STORE0 STORE1	"Store E"; data supplied by GSE is entered into E by GSE. Address E is also supplied by GSE.

TABLE 32-2
MACHINE INSTRUCTIONS (cont)

① Symbolic Instruction Word	② Order Code	③ Sub- instructions Executed	④ Description
PERIPHERAL INSTRUCTIONS (cont)			
INOTRD H	none	INOTRD	"In Out Read H"; displays c(H) on GSE. Channel address H is supplied by GSE.
INOTLD H	none	INOTLD	"In Out Load H"; data supplied by GSE is entered into H by GSE. Channel address H is supplied by GSE.

- ① Address symbol K can represent any address in the Central Processor (CP), E Memory or F Memory.
Address symbol F can represent an address in F Memory only.
Address symbol E can represent an address in the CP or E Memory only.
Address symbol H can represent any channel address.
Address symbol C can represent any counter address.
- ② Entered into SQ, or SQ and S.
- ③ The execution of each subinstruction, except DV0 and DV4, takes one MCT or about 11.7 μ sec. DV0 and DV4 together require 1 MCT.
- ④ Address symbol I represents address of instruction described.
Register symbols A, L, Q, B, S, SQ, and G refer to registers defined in table 30-1.
Expression c(K) means "content of location (or register) K".
- ⑤ Execution of these seven subinstructions takes only six MCT's.
- ⑥ The code which can be used with any K or E instruction, is contained in register S. Whenever address 0020, 0021, 0022, or 0023 is contained in register S, register G cycles or shifts the quantity it receives from a CP register before that quantity is transferred to one of the four locations (paragraph 30-41).

Table 32-3

MACHINE INSTRUCTIONS, ALPHABETICAL LISTING

Symbolic Instruction Word \triangle_1	Order Code \triangle_2	Name and Type
AD K	06.	"Add K"; an arithmetic instruction
ADS E	02.6	"Add to Storage E"; an arithmetic instruction
AUG E	12.4	"Augment E"; an arithmetic instruction
BZF F	11.2 11.4 11.6	"Branch on Zero to Fixed F"; a sequence changing instruction
BZMF F	16.2 16.4 16.6	"Branch on Zero or Minus to Fixed F"; a se- quence changing instruction
CA K	03.	"Clear and Add K" a fetching instruction
CAE E	03.	Alternate spelling of CA K when referring to E Memory
CAF F	03.	Alternate spelling of CA K when referring to F Memory
CCS E	01.0	"Count, Compare, and Skip on E"; a sequence changing instruction
COM	04.0000	"Complement"; CS A
CS K	04.	"Clear and Subtract K"; a fetching instruction
CYL	.0022	"Cycle Left"; a Special Instruction
CYR	.0020	"Cycle Right"; a Special Instruction
DAS E	02.0	"Double Add to Storage E"; an arithmetic instruction
DCA K	13.	"Double Clear and Add K"; a fetching instruc- tion
DCS K	14.	"Double Clear and Subtract K"; a fetching in- struction
DCOM	14.0000	"Double Precision Complement"; DCS A
DDOUBL	02.0000	"Double Precision Double"; DAS A

Table 32-3

MACHINE INSTRUCTIONS, ALPHABETICAL LISTING (cont)

Symbolic Instruction Word $\triangle 1$	Order Code $\triangle 2$	Name and Type
DIM E	12.6	"Diminish E"; an arithmetic instruction
DINC C	none	"Diminish Increment C"; a Counter Instruction
DOUBLE	06.0000	"Double"; AD A
DTCB	05.2005	"Double Precision Transfer Control Both Banks"; DXCH Z
DTCF	05.2004	"Double Precision Transfer Control Fixed Bank"; DXCH FBANK
DV E	11.0	"Divide by E"; an arithmetic instruction
DXCH E	05.2	"Double Exchange A and E"; a fetching and storing instruction
EDOP	.0023	"Edit Operator"; a Special Instruction
EXTEND	00.0006	"Extend"; a Special Instruction
FETCH K	none	"Fetch K"; a Peripheral Instruction
GO	00.	"GO"; an Interrupting Instruction
INCR E	02.4	"Increment E"; an arithmetic instruction
INDEX E	05.0	Alternate spelling for NDX E
INDEX K	15.	Alternate spelling for NDX K
INHINT	00.0004	"Inhibit Interrupt"; a Special Instruction
INOTLD H	none	"In Out Load H"; a Peripheral Instruction
INOTRD H	none	"In Out Read H"; a Peripheral Instruction
LXCH E	02.2	"Exchange L and E"; a fetching and storing instruction
MASK K	07.	Alternate spelling of MSK K
MCDU C	none	"Minus CDU C"; a Counter Instruction

Table 32-3

MACHINE INSTRUCTIONS, ALPHABETICAL LISTING (cont)

Symbolic Instruction Word $\triangle 1$	Order Code $\triangle 2$	Name and Type
MINC C	none	"Minus Increment C"; a Counter Instruction
MP K	17	"Multiply K"; an arithmetic instruction
MSK K	07.	"Mask with K"; a logic instruction
MSU E	12.0	"Modular Subtract E"; an arithmetic instruction
NDX E	05.0	"Index Next Basic Instruction with E"; a modifying instruction
NDX K	15.	"Index Next Extra-Code Instruction with K"; a modifying instruction
NOOP	03.0000	"No Operation (Erasable)"; instruction is stored in E Memory; CA A
NOOP	TCF(I+1)	"No Operation (Fixed)"; where I is address of instruction TCF (I+1) stored in F Memory
OVSK	05.4000	"Overflow Skip"; TS A
PCDU C	none	"Plus CDU C"; a Counter Instruction
PINC C	none	"Plus Increment C"; a Counter Instruction
QXCH E	12.2	"Exchange Q and E"; a fetching and storing instruction
RAND H	10.2	"Read and AND H"; a Channel Instruction
READ H	10.0	"Read H"; a Channel Instruction
RELINT	00.0003	"Release Interrupt Inhibit"; a Special Instruction
RESUME	05.0017	"Resume Interrupted Program"; a Special Instruction
RETURN	00.0002	"Return"; TC Q
ROR H	10.4	"Read and OR H"; a Channel Instruction

Table 32-3

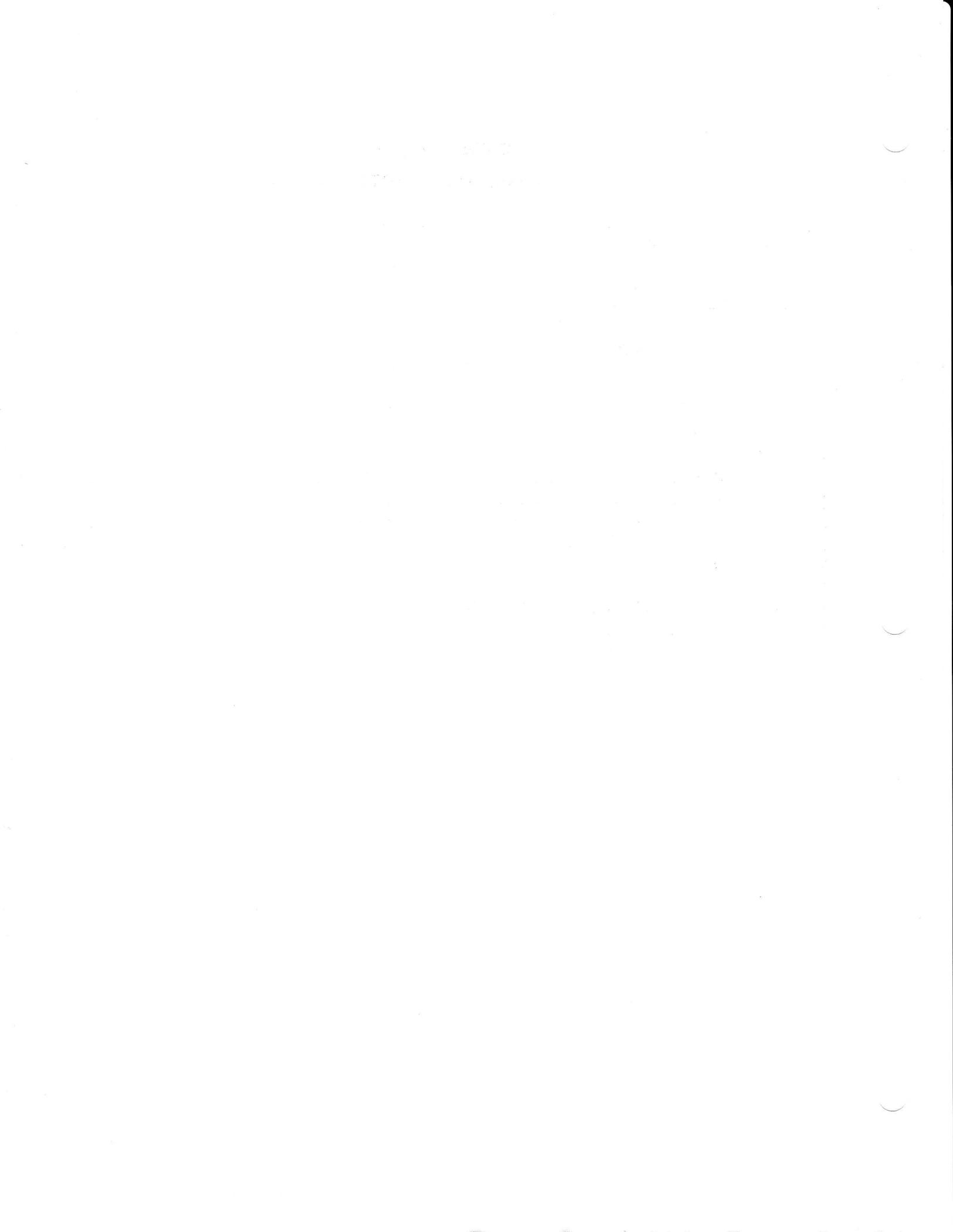
MACHINE INSTRUCTIONS, ALPHABETICAL LISTING (cont)

Symbolic Instruction Word $\triangle 1$	Order Code $\triangle 2$	Name and Type
RUPT	10.7	"Interrupt Program Execution"; an Interrupting Instruction
RXOR H	10.6	"Read and Exclusive OR H"; a Channel Instruction
SHANC C	none	"Shift and Add Increment C"; a Counter Instruction
SHINC C	none	"Shift Increment C"; a Counter Instruction
SQUARE	17.0000	"Square"; MP A
SR	.0021	"Shift Right"; a Special Instruction
STORE E	none	"Store E"; a Peripheral Instruction
SU E	16.0	"Subtract E"; an arithmetic instruction
TCAA	05.4005	"Transfer Control to Address in A"; TS Z
TC K	00.	"Transfer Control to K"; a sequence changing instruction
TCF F	01.2 01.4 01.6	"Transfer Control to Fixed F"; a sequence changing instruction
TCR K	00.	Alternate spelling of TC K (Transfer Control Setting up Return)
TCSAJ K	00.	"Transfer Control to Specified Address K"; a Peripheral Instruction
TS E	05.4	"Transfer to Storage E"; a storing instruction
WAND H	10.3	"Write and AND H"; a Channel Instruction
WOR H	10.5	"Write and OR H"; a Channel Instruction
WRITE H	10.1	"Write H"; a Channel Instruction
XCH E	05.6	"Exchange A and E"; a fetching and storing instruction

Table 32-3

MACHINE INSTRUCTIONS, ALPHABETICAL LISTING (cont)

Symbolic Instruction Word 	Order Code 	Name and Type
ZL	02. 2007	"Zero L"; LXCH ZERO
ZQ	12. 2007	"Zero Q"; QXCH ZERO
<p> Address symbol K can represent any address in the Central Processor (CP), E Memory or F Memory. Address symbol F can represent an address in F Memory only. Address symbol E can represent an address in the CP or E Memory only.</p> <p> Entered into SQ, or SQ and S.</p>		



32-3. EXECUTION OF INSTRUCTIONS

32-4. The execution of all Machine Instructions is under the control of the Sequence Generator (SQG). The initiation of instruction executions is described in paragraphs 30-24 through 30-27. All Machine Instructions are composed of one, two, three, or seven subinstructions, as indicated in the third column of table 32-2. All but two subinstructions consist of twelve actions. Refer to table 32-4 at the end of this section and paragraph 30-21. Subinstructions DV0 and DV4 together consist of twelve actions. An action is defined as a set of control pulses generated by the SQG and may be composed of zero, one, or several control pulses. One action occurs every 0.977 μ sec and the execution of one subinstruction takes 11.7 μ sec which equals one Memory Cycle Time (MCT).

32-5. EXECUTION OF SUBINSTRUCTIONS

32-6. When a Regular Instruction, instruction RUPT, instruction GO, or instruction TCSAJ K is executed, the content of register SQ and the content of the stage counter (ST) determine the subinstruction to be executed as shown in columns three and four of table 32-4. Subinstruction STD2 (standard two) is executed whenever the stage counter (ST) contains octal 2 regardless of the contents of register SQ as indicated by the X symbols. If the stage counter (ST) contains any other octal number than 2, a subinstruction is executed as defined by the content of register SQ. Subinstructions of Regular Instructions with whole order codes are determined by the content of bit positions EXT through 13 of register SQ while the content of bit positions 12 through 10 is irrelevant. Subinstructions of Regular Instructions with quarter codes are defined by the content of bit positions EXT through 11 while the content of bit position 10 is irrelevant. Subinstructions of Channel Instructions are defined by the content of bit positions EXT through 10. (Refer to paragraphs 30-153 and 30-154.)

32-7. When a Counter Instruction is executed, the contents of register SQ and the stage counter (ST) are irrelevant; the execution of Counter Instructions is determined by the setting of certain flip-flops only. When a Peripheral Instruction is executed, the setting of certain flip-flops and the content of the stage counter (ST) determine the subinstruction being executed.

32-8. The twelve actions (1 through 12) of a DV subinstruction do not occur in the same sequence as time pulses T01 through T12 are generated. Actions 1 through 3 of subinstruction DV0 are caused by time pulses 1 through 3.

Actions 4 through 12 and 1 through 3 of subinstructions DV1 through DV6 (table 30-4) are caused by time pulses 4 through 12, and 1 through 3 in that sequence. Actions 4 through 12 of subinstruction DV4 are caused by time pulses 4 through 12 and complete the last MCT. Thus, the execution of the six DV subinstructions takes only five MCT's.

32-9. CONTROL PULSES

32-10. Control pulses are signals generated by the SQG which regulates data flow within the Central Processor (CP) and the Input-Output Control. The control pulses can be grouped in five categories: read pulses, write pulses, direct read-write pulses, test pulses, and special pulses (paragraph 30-28). All control pulses are defined in table 32-5 at the end of this section.

32-11. A read pulse gates the content of a register or input-output channel into the write amplifiers (WA's). Read pulses such as RA, RB, etc., read the content of a specific register into the WA's. Read pulses RSC and RCH read the content of that CP register or input-output channel into the WA's the address of which is contained in register S. Read pulses R15, RIC, etc., enter certain octal quantities into the WA's.

32-12. A write pulse clears a register or input-output channel and gates into it the data which is present at the WA's, i. e., the data which is gated into the WA's by a read pulse at the same time. Write control pulses such as WA, WB, etc., write into a specific register. Write pulses WSC and WCH write into a register or channel which is defined by the content of register S.

32-13. Direct read-write pulses copy the content of one register into another register without using the WA's. Control pulse A2X, for example, enters the content of register A into register X.

32-14. Test control pulses test the content of certain bit positions, set the branch flip-flops accordingly and thus initiate branching operations. For instance, control pulse TSGN tests the content of WA 16 (bit 16) and sets flip-flop BR1 to ONE if WA 16 contains a ONE (minus sign).

32-15. Special control pulses are used to set stage counters, certain flip-flops, to initiate certain operations, etc.

32-16. SUBINSTRUCTION STD2

32-17. Subinstruction STD2 (standard two) is used as a concluding subinstruction with most Regular Instructions and instructions RUPT and TCSAJ K. Its purpose is to increment by one the content of register Z, the program

counter, and to call forward the instruction to be executed next. Subinstruction STD2 is executed when the stage counter (ST) contains 2.

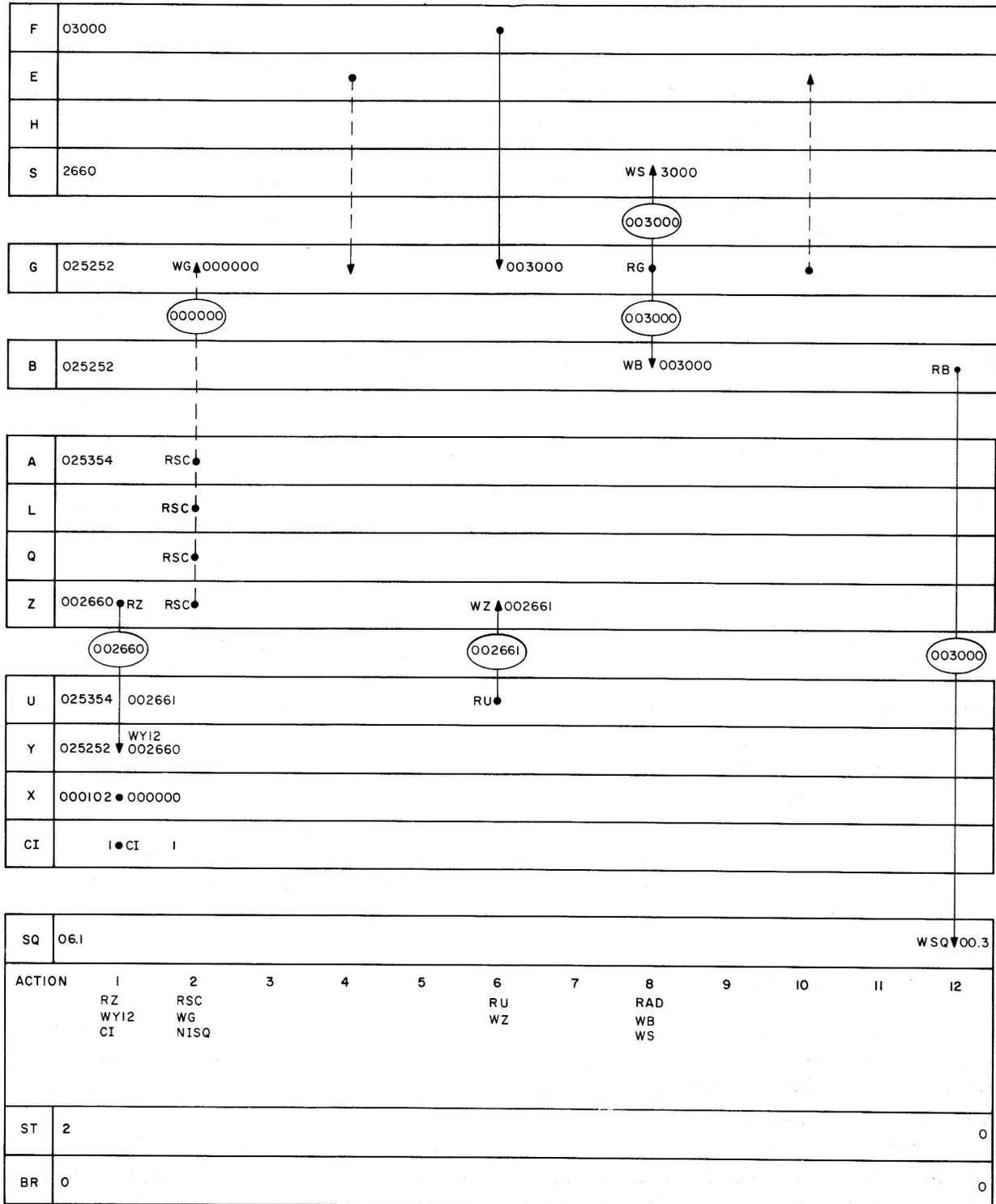
32-18. Control pulses RZ and WY12 of action 1 (row 1 of table 32-4) clear the Adder and enter bits 12 through 1 contained in register Z into Adder input register Y. Pulse CI enters a carry bit into bit position 1 of the Adder, thus adding a one to the quantity entered by pulse WY12. The incremented quantity is returned to register Z by pulses RU and WZ of action 6. Before this operation, register Z contained the address of the instruction to be executed next. After the operation, register Z contains the address of the instruction to be executed thereafter which becomes the "address of the next instruction" during the execution of the next instruction.

32-19. Control pulses RSC and WG of action 2 clear register G if no CP register address (addresses 0000 through 0007) is contained in register S. If register S contains a CP register address, the content of the specified CP register is entered into register G. If register S contains an E memory address (address 0010 through 1777), the content of the specified E location is entered automatically into register G at time 4 by the E Memory (paragraph 30-52). If register S contains an F memory address (addresses 2000 and above), the content of the specified F location is entered automatically into register G at time 6 by the F Memory (paragraph 30-53). The information being entered into register G is the instruction to be executed after the current STD2 subinstruction. The content of register G is returned automatically to an E memory location after time 10 if an E memory location is addressed to restore the content of the location (destroyed during readout).

32-20. Control pulse RAD of action 8 normally generates control pulse RG. Pulses RG, WB, and WS of action 8 enter the next instruction into register B and its relevant address into register S. Control pulse NISQ of action 2, causes the generation of pulses RB and WSQ at time 12, thus entering bits 15 through 10 of the next instruction into register SQ and initiating the execution of the next instruction. See example in paragraphs 32-25 through 32-29.

32-21. DATA TRANSFER DIAGRAMS

32-22. The data transfer diagrams are used to describe the operation of subinstructions. Figure 32-1, for instance, illustrates the execution of subinstruction STD2 discussed in paragraphs 32-16 through 32-20. Box F at the top represents a location in F Memory if one has been addressed, box E, a location in E Memory if one has been addressed, and box H an input-output channel if one has been addressed. The subsequent boxes represent CP registers and the Adder with input registers Y and X, output gates (U), and carry input flip-flop CI. The large box below register SQ represents the SQG. The control pulses generated at the various actions are listed in this box.



2700A

Figure 32-1. Subinstruction STD2

The 3-bit stage counter (ST) and the 2-bit branch flip-flops (BR) are represented by two small boxes at the bottom. (Branch flip-flop BR1 contains the high order bit and BR2, the low order bit.) Data shown in the registers prior to action 1 indicate starting conditions.

32-23. The information flow caused by the control pulses is indicated by vertical lines. Numbers in ellipses indicate data passing through the WA's. Information moving between memory and register G does not pass through the WA's, therefore, no ellipses are shown in the respective flow lines. When data is gated directly from one register into another, no flow line is shown. Broken flow lines are used to indicate information flow which may occur under conditions different from those pertaining to the given numeric example.

32-24. The 5-digit octal quantities used in boxes F, E, and CH represent 15 bit words (bits 15 through 1, no parity bit). Register S is able to store 12 bit addresses represented by 4 octal digits. Registers G, B, A, L, Q, Z, Y, and X are able to store 16 bit words represented by 6 octal digits and the same is true for the output gates (U) of the Adder. Register SQ is able to store 7 bits expressed in fractional octal numbers, 4 bits or 2 octal digits in front of the octal point, and 3 bits or 1 octal digit after the octal point.

32-25. EXAMPLE OF INSTRUCTION EXECUTIONS

32-26. The following sequence of instructions has been chosen as an example:

Location	Instruction	Code
2657	AD 1213	6.1213
2660	TC 3000	0.3000
2661	CS 2765	4.2765
3000	CA 0375	3.0375

Let us assume that subinstruction AD0 has been executed and that subinstruction STD2 is being executed. Prior to time 1 of STD2, register Z and S contain 2660, the address of instruction TC 3000 to be executed next, as indicated in figure 32-1. Since address 2660 refers to F Memory, instruction TC 3000 (03000) is shown in the top box. The contents of registers G, B, A, Y, X, and SQ, which remained from the execution of subinstruction AD0, are irrelevant. The stage counter has been set to 2 at the last time 12 to initiate the execution of subinstruction STD2.

32-27. Control pulse RZ of action 1 gates address 002660 into the WA's and pulse WY12 gates the content of WA's 12 through 1 into register Y. Pulse WY12 also clears register X. Pulse CI forces a carry bit into bit position 1

of the Adder and the quantity 002661 appears at the output gates (U). Pulses RU and WZ of action 6 return the incremented address to register Z.

32-28. Since register S does not contain a CP register address, no CP register is gated for read out at the occurrence of pulse RSC of action 2, the WA's contain 000000, and pulse WG writes this quantity into register G, thus clearing register G. Since register S also does not contain an E memory address, nothing is entered into register G by E Memory at time 4. Because register S contains address 2660 of F Memory, instruction TC 3000 contained at location 2660 is entered into register G by F Memory at time 6. Since register S does not contain an E memory address prior to time 6, no data is restored in E Memory after time 10.

32-29. Control pulse RAD of action 8 is interpreted as RG and enters instruction TC 3000 into the WA's. Pulse WB enters the same instruction into register B while pulse WS enters the relevant address into register S. Control pulses RB and WSQ, which are caused by pulse NISQ, enter 00.3 into register SQ at time 12. The stage counter is reset to 0. Thus the execution of subinstruction TC0 has been initiated.

(text continued on page 32-53)

TABLE 32-4
CONTROL PULSES GENERATED AT VARIOUS ACTIONS (cont)

Row No.	Subinstruction Symbol	c(SQ)			c(ST) 3,2,1	Actions												Remarks	
		EXT	16,14,13	12,11,10		1	2	3	4	5	6	7	8	9	10	11	12		
6	BZMF0	1 1 1	110 110 110	01X 10X 11X	000 000 000	RA WG TSGN TMZ	TPZG	RSC WG		$\triangle 1$ - $\triangle 2$ } $\triangle 3$ } RB $\triangle 4$ } WY12 CI	$\triangle 1$ - $\triangle 2$ } $\triangle 3$ } RU $\triangle 4$ } WZ		$\triangle 1$ RZ WS ST2 $\triangle 2$ } $\triangle 3$ } RAD WB WS $\triangle 4$ } NISQ						$\triangle 1$ If c(BR) = 0, c(A) is positive non-zero at times 1 and 2. $\triangle 2$ If c(BR) = 1, c(A) is plus zero at times 1 and 2. $\triangle 3$ If c(BR) = 2, c(A) is negative non-zero at times 1 and 2. $\triangle 4$ If c(BR) = 3, c(A) is minus zero at times 1 and 2. If c(BR) = 0, BZMF0 is followed by STD2. If c(BR) \neq 0, BZMF0 is followed by instruction to which control is transferred.
7	CA0	0	011	XXX	000		RSC WG					RG WB	RZ WS ST2	RB WG	RB WA			Followed by STD2.	
8	CS0	0	100	XXX	000		RSC WG					RG WB	RZ WS ST2	RB WG	RC WA			Followed by STD2.	
9	DCA0	1	011	XXX	000	RB WY12 MONEX CI	RSC WG					RG WB	RU WS	RB WG	RB WL ST1			Followed by DCA1.	
10	DCA1	1	011	XXX	001		RSC WG					RG WB	RZ WS ST2	RB WG	RB WA			Followed by STD2.	
11	DCS0	1	100	XXX	000	RB WY12 MONEX CI	RSC WG					RG WB	RU WS	RB WG	RC WL ST1			Followed by DCS1.	
12	DCS1	1	100	XXX	001		RSC WG					RG WB	RZ WS ST2	RB WG	RC WA			Followed by STD2.	

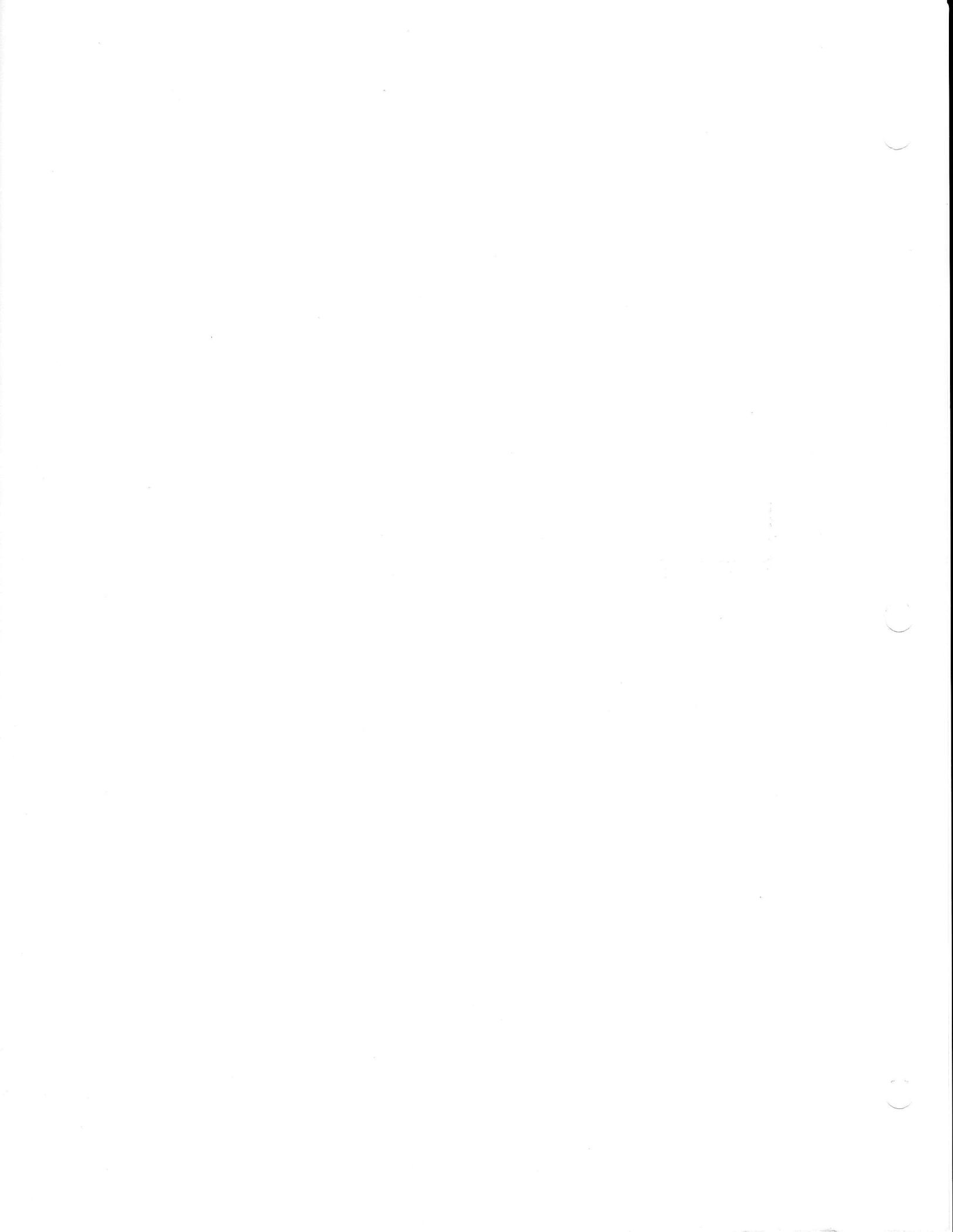


TABLE 32-4
CONTROL PULSES GENERATED AT VARIOUS ACTIONS (cont)

Row No.	Subinstruction Symbol	c(SQ)			c(ST)	Actions												Remarks	
		EXT	16,14,13	12,11,10		3,2,1	1	2	3	4	5	6	7	8	9	10	11		12
13	TS0	0	1 0 1	1 0 X	0 0 0	RL10BB WS	RSC WG	RA WB TOV	① RZ WY12 ② RZ WY12 CI ③ RZ WY12 CI	① — ② RB1 WA ③ R1C WA	RU WZ	RB WSC WG	RZ WS ST2						① If c(BR) = 0, A16,15 = 0 or 3 at time 3. ② If c(BR) = 1, A16,15 = 1 at time 3. ③ If c(BR) = 2, A16,15 = 2 at time 3. Followed by STD2.
14	XCH0	0	1 0 1	1 1 X	0 0 0	RL10BB WS	RSC WG	RA WB		RG WA		RB WSC WG	RZ WS ST2						Followed by STD2.
15	LXCH0	0	0 1 0	0 1 X	0 0 0	RL10BB WS	RSC WG	RL WB		RG WL		RB WSC WG	RZ WS ST2						Followed by STD2.
16	QXCH0	1	0 1 0	0 1 X	0 0 0	RL10BB WS	RSC WG	RQ WB		RG WQ		RB WSC WG	RZ WS ST2						Followed by STD2.
17	DXCH0	0	1 0 1	0 1 X	0 0 0	RL10BB WS WY12 MONEX CI	RSC WG	RL WB		RG WL		RB WSC WG	RU WS WB		ST1				Control pulse CI at time 1 causes 000001 plus 177776 to result in 000000 instead of 177777. Followed by DXCH1.
18	DXCH1	0	1 0 1	0 1 X	0 0 1	RL10BB WS	RSC WG	RSC WB		RG WA		RB WSC WG	RZ WS ST2						Followed by STD2.
19	NDX0	0	1 0 1	0 0 X	0 0 0		RSC WG			TRSM		RG WB	RZ WS	RB WG	ST1				Normally followed by NDX1. Followed by RSM3 if c(S) = 0017 at time 5.
20	NDX1	0	1 0 1	0 0 X	0 0 1	RZ WY12 CI	RSC WG NISQ	RB WZ	RA WB	RZ WA	RU WZ	RG WY A2X	RU WS	RB WA	RU WB				Followed by indexed Basic Instruction. See note A at end of table.
21	NDXX0	1	1 0 1	X X X	0 0 0		RSC WG					RG WB	RZ WS	RB WG	ST1				Followed by NDXX1.

TABLE 32-4
CONTROL PULSES GENERATED AT VARIOUS ACTIONS (cont)

Row No.	Subinstruction Symbol	c(SQ)			c(ST)	Actions												Remarks
		EXT	16,14,13	12,11,10		3,2,1	1	2	3	4	5	6	7	8	9	10	11	
22	NDDX1	1	1 0 1	X X X	0 0 1	RZ WY12 CI	RSC WG NISQ	RB WZ	RA WB	RZ WA	RU WZ	RG WY A2X	RU WS	RB WA	RU WB EXT		Followed by an indexed Extra Code Instruction.	
23	ADD	0	1 1 0	X X X	0 0 0		RSC WG					RG WB	RZ WS ST2	RB WG	RB WY A2X	RU WA	Followed by STD2.	
24	SU0	1	1 1 0	0 0 X	0 0 0		RSC WG					RG WB	RZ WS ST2	RB WG	RC WY A2X	RU WA	Followed by STD2.	
25	MP0	1	1 1 1	X X X	0 0 0		RSC WG	RA WB TSGN	△1 RB WL △2 RC WL			RG WB TSGN2	RZ WS	△3 RB WY △4 RB WY CI △5 RC WY CI △6 RC WY	RU WB TSGN NEACON ST1	△7 WA △8 RB1 R1C WA L16	△1 If c(BR) = 0 or 1, A16 = 0 at time 3. △2 If c(BR) = 2 or 3, A16 = 1 at time 3. △3 If c(BR) = 0, A16 = 0 and G16 = 0 at time 7. △4 If c(BR) = 1, A16 = 0 and G16 = 1 at time 7. △5 If c(BR) = 2, A16 = 1 and G16 = 0 at time 7. △6 If c(BR) = 3, A16 = 1 and G16 = 1 at time 7. △7 If c(BR) = 0 or 1, U16 = 0 at time 10. △8 If c(BR) = 2 or 3, U16 = 1 at time 10. Followed by MP1.	
26	MP1	1	1 1 1	X X X	0 0 1	ZIP	ZAP	ZIP	ZAP	ZIP	ZAP	ZIP	ZAP	ZIP	ZAP ST1 ST2	ZIP	Followed by MP3.	
27	MP3	1	1 1 1	X X X	0 1 1	ZAP	ZIP NISQ	ZAP	RSC WG	RZ WY12 CI	RU WZ TL15 NEACOF	△1 — △2 RB WY A2X	RAD WB WS	RA	RL	△1 — △2 RU WA	△1 If c(BR) = 0 or 1, L15 = 0 at time 6. △2 If c(BR) = 2 or 3, L15 = 1 at time 6. Followed by next instruction. See note A.	

TABLE 32-4
CONTROL PULSES GENERATED AT VARIOUS ACTIONS (cont)

Row No.	Subinstruction Symbol	c(SQ)			c(ST)	Actions												Remarks		
		EXT	16,14,13	12,11,10		3,2,1	1	2	3	4	5	6	7	8	9	10	11		12	
28	DV0	1	001	00X	000	RA WB TSGN TMZ	① RC WA TMZ ② DVST DVST	RU WB STAGE												① If c(BR) = 0 or 1, A16 = 0 at time 2. ② If c(BR) = 2 or 3, A16 = 1 at time 2. Followed by DV1 after action 3.
29	DV1	1	001	00X	001				① RL WB ② RL WB TSGN	③ RB WY B15X ④ RC WY B15X Z16	RU WL TOV	RG RSC WB TSGN	⑤ RA WY PONEX ⑥ —	⑦ RB WA RU WB	RL WYD	RU WL			① If c(BR) = 0 or 2, c(A) at time 2 of DV0 is non-zero. ② If c(BR) = 1 or 3, c(A) at time 2 of DV0 is minus zero. ③ If c(BR) = 0 or 1, A16 = 0 at time 2 of DV0 or L16 = 0 at time 4 of DV1. ④ If c(BR) = 2 or 3, A16 = 1 at time 2 of DV0 or L16 = 0 at time 4 of DV1. ⑤ If c(BR) = 0 or 2, no overflow at time 8. ⑥ If c(BR) = 1 or 3, overflow at time 8. ⑦ If c(BR) = 0 or 1, G16 = 0 at time 9. ⑧ If c(BR) = 2 or 3, G16 = 1 at time 9. ⑨ If c(BR) = 0 or 1, U16 = 0 at time 2. ⑩ If c(BR) = 2 or 3, U16 = 1 at time 2. Followed by DV3 after action 3.	
29 (cont)	DV1	1	001	00X	001	L2GD RB WYD A2X PIFL	RG WL TSGU DVST ⑨ CLXC ⑩ RBIF	RU WB STAGE												
30	DV3	1	001	00X	011				L2GD RB WYD A2X PIFL	RG WL TSGU CLXC ① CLXC ② RBIF	RU WB	L2GD RB WYD A2X PIFL	RG WL TSGU CLXC ① CLXC ② RBIF	RU WB	L2GD RB WYD A2X PIFL	RG WL TSGU CLXC ① CLXC ② RBIF	RU WB		① If c(BR) = 0 or 1 at time 5, 8, or 11, U16 = 0. ② If c(BR) = 2 or 3 at time 5, 8, or 11, U16 = 1.	
30 (cont)	DV3	1	001	00X	011	L2GD RB WYD A2X PIFL	RG WL TSGU DVST CLXC ③ CLXC ④ RBIF	RU WB STAGE												③ If c(BR) = 0 or 1, U16 = 0 at time 2. ④ If c(BR) = 2 or 3, U16 = 1 at time 2. Followed by DV7 after action 3.

TABLE 32-4
CONTROL PULSES GENERATED AT VARIOUS ACTIONS (cont)

Row No.	Subinstruction Symbol	c(SQ)			c(ST)	Actions												Remarks	
		EXT	16,14,13	12,11,10	3,2,1	1	2	3	4	5	6	7	8	9	10	11	12		
31	DV7	1	001	00X	111				L2GD RB WYD A2X PIFL	RG WL TSGU CLXC RBLF	RU WB	L2GD RB WYD A2X PIFL	RG WL TSGU CLXC RBLF	RU WB	L2GD RB WYD A2X PIFL	RG WL TSGU CLXC RBLF	RU WB		
31 (cont)	DV7	1	001	00X	111	L2GD RB WYD A2X PIFL	RG WL TSGU DVST CLXC RBLF	RU WB STAGE											
32	DV6	1	001	00X	110				L2GD RB WYD A2X PIFL	RG WL TSGU CLXC RBLF	RU WB	L2GD RB WYD A2X PIFL	RG WL TSGU CLXC RBLF	RU WB	L2GD RB WYD A2X PIFL	RG WL TSGU CLXC RBLF	RU WB		
32 (cont)	DV6	1	001	00X	110	L2GD RB WYD A2X PIFL	RG WL TSGU DVST CLXC RBLF	RU WB STAGE											
33	DV4	1	011	00X	100				L2GD RB WYD A2X PIFL	RG WB WA TSGU CLXC RBLF	RZ TOV	RC WA RC WA	RZ WS ST2 TSGN RSTSTG	RU WB WL	RC WL				

1 If c(BR) = 0 or 1, U16 = 0 at time 5, 8, 11, or 2.
 2 If c(BR) = 2 or 3, U16 = 1 at time 5, 8, 11, or 2.
 Followed by DV6 after action 3.
 1 If c(BR) = 0 or 1, U16 = 0 at time 5, 8, 11, or 2.
 2 If c(BR) = 2 or 3, U16 = 1 at time 5, 8, 11, or 2.
 Followed by DV4 after action 3.
 1 If c(BR) = 0 or 1 at time 5, U16 = 0.
 2 If c(BR) = 2 or 3 at time 5, U16 = 1.
 3 If c(BR) = 0 at time 7, U16,15 = 00 or 11.
 4 If c(BR) = 1 at time 7, U16,15 = 01.
 5 If c(BR) = 2 at time 7, U16,15 = 10.
 6 If c(BR) = 0 or 1 at time 10, Z16 = 0.
 7 If c(BR) = 2 or 3 at time 10, Z16 = 1.
 Followed by STD2 after time 12.

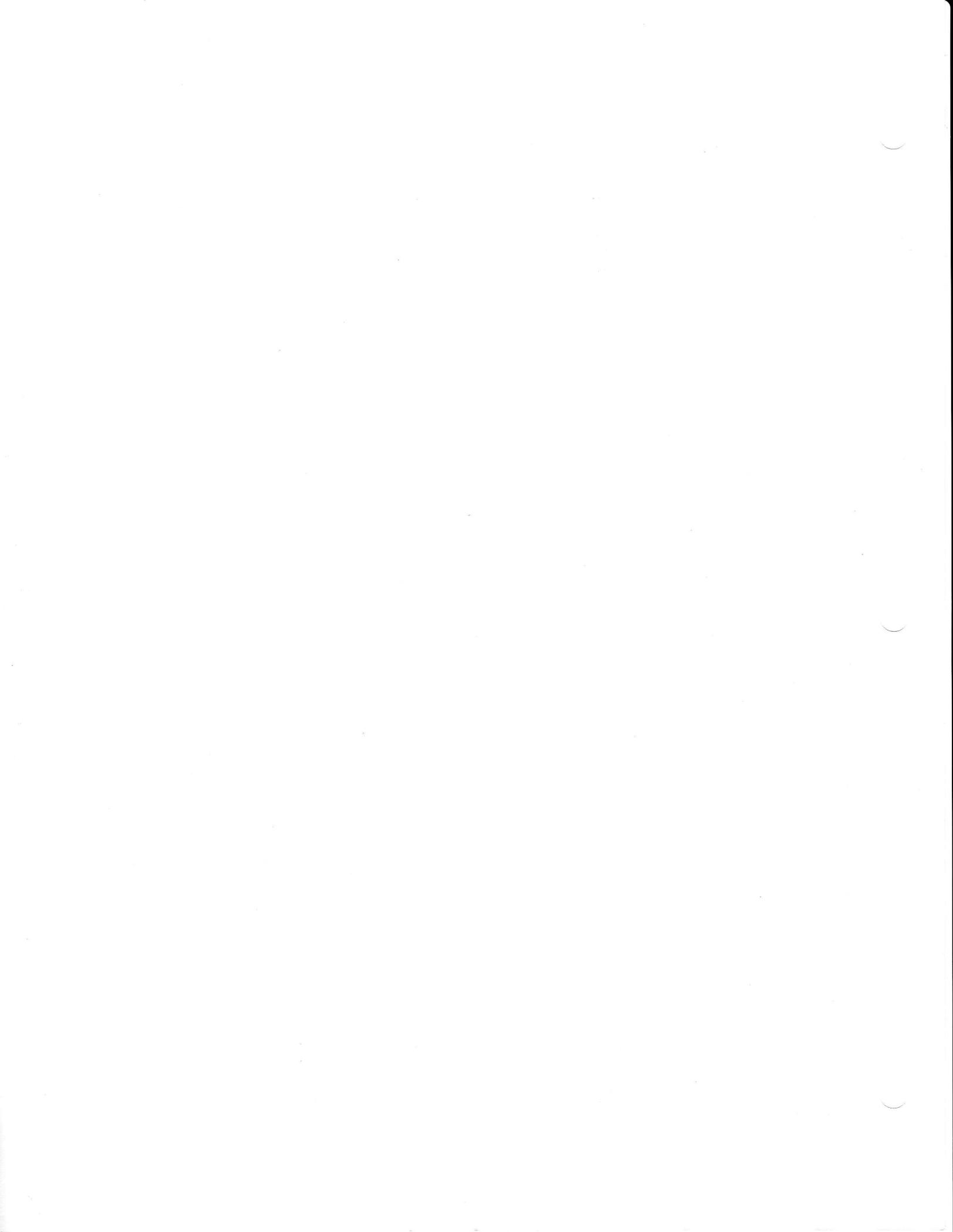


TABLE 32-4
CONTROL PULSES GENERATED AT VARIOUS ACTIONS (cont)

Row No.	Subinstruction Symbol	c(SQ)			c(ST)	Actions												Remarks
		EXT	16,14,13	12,11,10	3,2,1	1	2	3	4	5	6	7	8	9	10	11	12	
34	ADSO	0	010	11X	000	RL10BB WS	RSC WG			RG WY A2X	RU WSC WG TOV	① WA ② WA RBI ③ WA RIC	RZ WS ST2	RC TMZ		RU WA		① If c(BR) = 0, U16,15 = 00 or 11 at time 7. ② If c(BR) = 1, U16,15 = 01 at time 7. ③ If c(BR) = 2, U16,15 = 10 at time 7; RC and TMZ at time 9 have no effect. Followed by STD2.
35	DASO	0	010	00X	000	RL10BB WS WY12 MONEX CI	RSC WG	RA WB	RL WA	RU WL	RG WY A2X	RB WA	RL WB	RU WSC WG TOV	① RA WY ST1 ② RA WY PONEX ST1 ③ RA WY MONEX ST1			① If c(BR) = 0, U16,15 = 00 or 11 at time 9. ② If c(BR) = 1, U16,15 = 01 at time 9. ③ If c(BR) = 2, U16,15 = 10 at time 9. Followed by DAS1.
36	DASI	0	010	00X	001	RL10BB WS	RSC WG	RU WA		RG WY A2X	RU WG WSC TOV	① WA ② WA RBI ③ WA RIC	RZ WS ST2	RC TMZ	④ WL ⑤ —	④ — ⑤ RU WA		① If c(BR) = 0, U16,15 = 00 or 11 at time 7. ② If c(BR) = 1, U16,15 = 01 at time 7. ③ If c(BR) = 2, U16,15 = 10 at time 7. ④ If c(BR) = 0 or 2, $\bar{c}(B) \neq 17777$ at time 9. ⑤ If c(BR) = 1 or 3, $\bar{c}(B) = 17777$ at time 9. Followed by STD2.
37	INCRO	0	010	10X	000	RL10BB WS	RSC WG			RG WY TSGN TMZ TPZG	PONEX	RU WSC WG WOVR	RZ WS ST2					TSGN, TMZ, and TPZG of action 5 have no effect. If U16,15 = 01, WOVR of action 7 requests execution of PINC 0024 if c(S) = 0025, or RUPT if c(S) = 0026, 0027, or 0030. Followed by STD2.

TABLE 32-4
CONTROL PULSES GENERATED AT VARIOUS ACTIONS (cont)

Row No.	Subinstruction Symbol	c(SQ)			c(ST)	Actions												Remarks	
		EXT	16,14,13	12,11,10	3,2,1	1	2	3	4	5	6	7	8	9	10	11	12		
38	AUGO	1	0 1 0	1 0 X	0 0 0	RL10BB WS	RSC WG			RG WY TSGN TMZ TPZG	① PONEX ② MONEX	RU WSC WG WOVR	RZ WS ST2						① If c(BR) = 0 or 1, G16 = 0 at time 6. ② If c(BR) = 2 or 3, G16 = 1 at time 6. TMZ and TPZG of action 5 have no effect. If U16, I5 = 11, WOVR of action 7 requests execution of PINC 0024 if c(S) = 0025 or RUPT if c(S) = 0026, 0027, or 0030. Followed by STD2.
39	DIMO	1	0 1 0	1 1 X	0 0 0	RL10BB WS	RSC WG			RG WY TSGN TMZ TPZG	① MONEX ② PONEX ③ —	RU WSC WG WOVR	RZ WS ST2					① If c(BR) = 0, c(G) is positive non-zero at time 6. ② If c(BR) = 2, c(G) is negative non-zero at time 6. ③ If c(BR) = 1 or 3, c(G) is plus or minus zero at time 6. WOVR at time 7 has no effect. Followed by STD2.	
40	MSUO	1	0 1 0	0 0 X	0 0 0	RL10BB WS	RSC WG			RG WB	RC WY CI A2X	RUS WA TSGN	RZ WS ST2	RB WG	① — ② RA WY MONEX	RUS WA		① If c(BR) = 0 or 1, U15 = 0 at time 7. ② If c(BR) = 2 or 3, U15 = 1 at time 7. Followed by STD2.	
41	MSKO	0	1 1 1	X X X	0 0 0		RSC WG	RA WB	RC WA			RG WB	RZ WS ST2	RC RA WY	RU WB	RC WA		Followed by STD2.	
42	READO	1	0 0 0	0 0 0	0 0 0	RL10BB WS	RA WB	WY	RCH WB	RB WA	RA WB		RZ WS ST2					Followed by STD2.	
43	WRITEO	1	0 0 0	0 0 1	0 0 0	RL10BB WS	RA WB WG	WY	RCH WB	RA WCH	RA WB		RZ WS ST2					See note B. Followed by STD2.	
44	RANDO	1	0 0 0	0 1 0	0 0 0	RL10BB WS	RA WB	RC WY	RCH WB	RC RU WA	RA WB	RC WA	RZ WS ST2					Followed by STD2.	

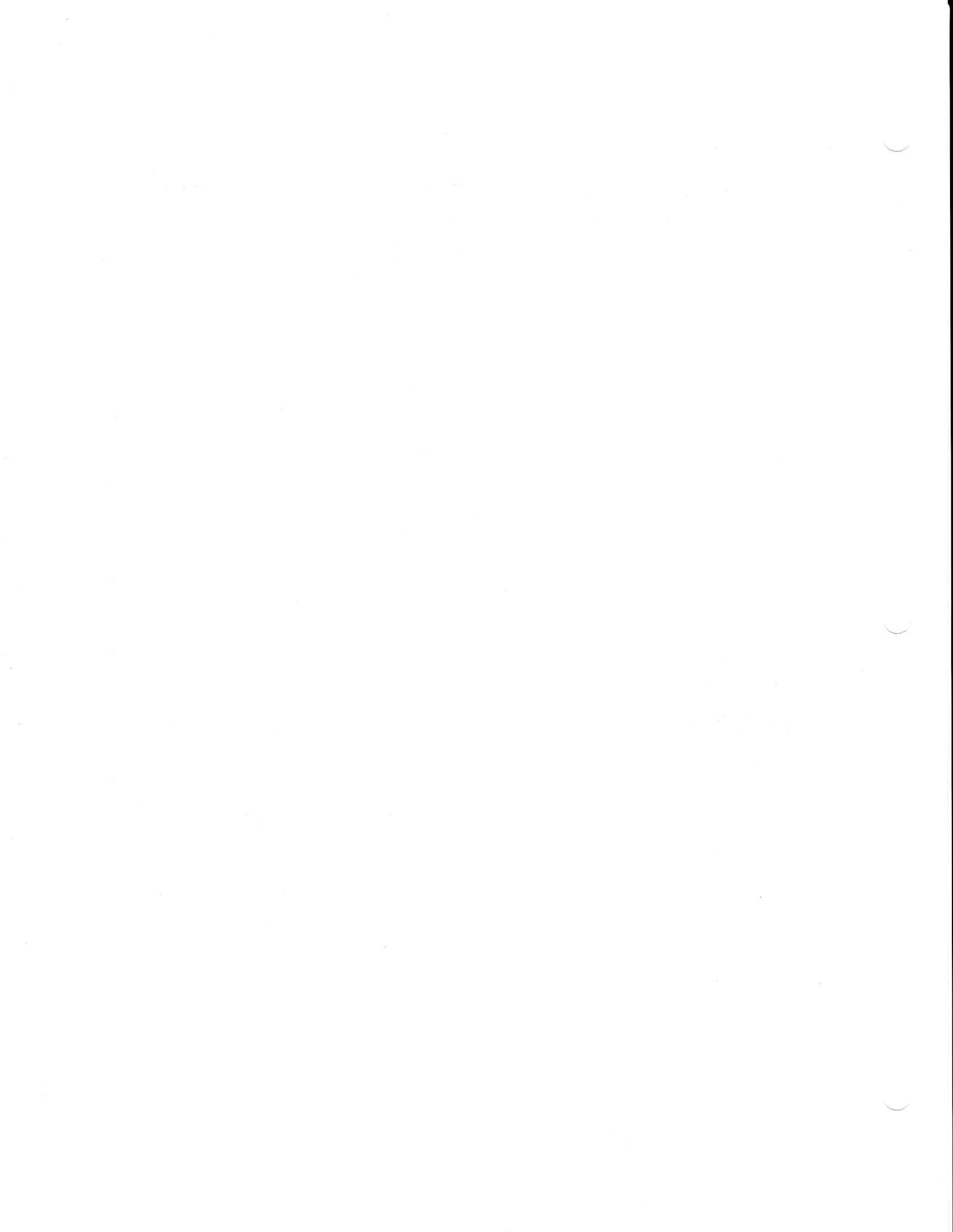


TABLE 32-4
CONTROL PULSES GENERATED AT VARIOUS ACTIONS (cont)

Row No.	Subinstruction Symbol	c(SQ)			c(ST)	Actions												Remarks
		EXT	16,14,13	12,11,10	3,2,1	1	2	3	4	5	6	7	8	9	10	11	12	
45	WAND0	1	000	011	000	RL10BB WS	RA WB	RC WY	RCH WB	RC RU WA	RA WB	RC WA WCH	RZ WS ST2					See note B. Followed by STD2.
46	ROR0	1	000	100	000	RL10BB WS	RA WB	RB WY	RCH WB	RB RU WA	RA WB		RZ WS ST2					Followed by STD2.
47	WOR0	1	000	101	000	RL10BB WS	RA WB	RB WY	RCH WB	RB RU WA WCH	RA WB		RZ WS ST2					See note B, Followed by STD2.
48	RXOR0	1	000	110	000	RL10BB WS	RA WB	RC RCH WY	RCH WB	RA RC WG		RG WB	RZ WS ST2	RC WG	RU WB	RC RG WA		Followed by STD2.
49	RSM3	0	101	00X	011	R15 WS	RSC WG NISQ			RG WZ	RB WG		RAD WB WS					Followed by instruction at return address. See note A.
50	RUPT0	1	000	111	000	R15 WS	RSC WG							RZ WG	ST1			RSC at action 2 has no effect. Followed by STD2.
51	RUPT1	1	000	111	001	R15 RB2 WS	RSC WG	RRPA WZ					RZ WS ST2	RB WG KRPT				RSC at action 2 has no effect. Followed by STD2.
52	GOJ1	0	000	XXX	001		RSC WG						RSTRT WS WB					Initiated by signal GOJAM. RSC at action 2 has no effect. Followed by TC 4000.
53	PINC	X	XXX	XXX	XXX	RSCT WS	RSC WG			RG WY TSGN TMZ TPZG	PONEX	RU WSC WG WOVR	RB WS					See note C. RSC of action 2, TSGN, TMZ, and TPZG of action 5, and WSC of action 7 have no effect. If U16,15 = 01, WOVR of action 7 requests the execution of PINC 0024 if c(S) = 0025, or RUPT if c(S) = 0026, 0027, or 0030.

TABLE 32-4
CONTROL PULSES GENERATED AT VARIOUS ACTIONS (cont)

Row No.	Subinstruction Symbol	c(SQ)			c(ST)	Actions												Remarks
		EXT	16,14,13	12,11,10	3,2,1	1	2	3	4	5	6	7	8	9	10	11	12	
54	MINC	X	X X X	X X X	X X X	RSCT WS	RSC WG			RG WY TSGN TMZ TPZG	MONEX	RU WSC WG WOVR	RB WS					See note C. RSC of action 2, TSGN, TMZ, and TPZG of action 5, and WSC and WOVR of action 7 have no effect.
55	DINC	X	X X X	X X X	X X X	RSCT WS	RSC WG			RG WY TSGN TMZ TPZG	① MONEX POUT ② PONEX MOUT ③ ZOUT	RU WSC WG WOVR	RB WS					See note C. ① If c(BR) = 0, c(G) is positive non-zero. ② If c(BR) = 2, c(G) is negative non-zero. ③ If c(BR) = 1 or 3, c(G) is plus or minus zero. RSC of action 2, and WSC and WOVR of action 7 have no effect.
56	PCDU	X	X X X	X X X	X X X	RSCT WS	RSC WG			RG WY TSGN TMZ TPZG	CI	RUS WSC WG WOVR	RB WS					See note C. RSC of action 2, TSGN, TMZ, and TPZG of action 5, and WSC and WOVR of action 7 have no effect.
57	MCDU	X	X X X	X X X	X X X	RSCT WS	RSC WG			RG WY TSGN TMZ TPZG	MONEX CI	RUS WSC WG WOVR	RB WS					See note C. RSC of action 2, TSGN, TMZ, and TPZG of action 5, and WSC and WOVR of action 7 have no effect.
58	SHINC	X	X X X	X X X	X X X	RSCT WS	RSC WG			RG WYD TSGN		RUS WSC WG WOVR	RB WS					See note C. RSC of action 2, and WSC and WOVR of action 7 have no effect. TSGN of action 5 requests RUPT if c(S) is 0045 and G16 = 1.
59	SHANC	X	X X X	X X X	X X X	RSCT WS	RSC WG			RG WYD TSGN CI		RUS WSC WG WOVR	RB WS					See note C. RSC of action 2, and WSC and WOVR of action 7 have no effect. TSGN of action 5 requests RUPT if c(S) is 0045 and G16 = 1.
60	TCSAJ3	0	0 0 0	0 0 0	0 1 1		RSC WG						WS WZ ST2					See note D. An address is sent to the WA's from an external source at time 8. Followed by STD2.

TABLE 32-4
CONTROL PULSES GENERATED AT VARIOUS ACTIONS (cont)

Row No.	Subinstruction Symbol	c(SQ)			c(ST)	Actions												Remarks
		EXT	16,14,13	12,11,10	3,2,1	1	2	3	4	5	6	7	8	9	10	11	12	
61	FETCH0	X	X X X	X X X	0 0 0	R6 WS	RSC WG WY ST1		WSC				WS					See note D. A bank number is sent to the WA's from an external source at time 4; a memory address is sent to the WA's at time 8. Followed by FETCH1.
62	FETCH1	X	X X X	X X X	0 0 1		RSC WG					RG	RB WS U2BBK		RBBK			See note D. The quantity contained in the WA's at time 7 and/or time 10 can be displayed on external equipment.
63	STORE0	X	X X X	X X X	0 0 0	R6 WS	RSC WG WY ST1		WSC				WS					See note D. A bank number is sent to the WA's from an external source at time 4; a memory address is sent to the WA's at time 8. Followed by STORE1.
64	STORE1	X	X X X	X X X	0 0 1		RSC WG		WSC			RG	RB WS U2BBK	WG	RBBK			See note D. A quantity is provided for loading from an external source at times 4 and 9.
65	INOTRD	X	X X X	X X X	X X X	WS	RSC WG			RCH			RB WS					See note D. A channel address is sent to the WA's from an external source at time 1; the quantity contained in the WA's at time 5 can be displayed on external equipment.
66	INOTLD	X	X X X	X X X	X X X	WS	RSC WG			RCH		WCH	RB WS					See note D. A channel address is sent to the WA's from an external source at time 1, and a quantity is sent at time 7.

NOTES:

- A. If c(G) = 000003 (RELINT), 000004 (INHINT), or 000006 (EXTEND), control pulse RAD causes the generation of control pulses RZ and ST2, and subinstruction STD2 is executed next. If G contains any other quantity, control pulse RAD causes the generation of control pulse RG and the next instruction is executed.
- B. The ONE entered into bit position 10 of register S has no effect on addressing channel locations.
- C. Counter Instructions are executed after any time 12 provided the execution of an interrupting or Peripheral Instruction is not being requested. Each Counter Instruction delays program execution for one MCT.
- D. Peripheral Instructions are initiated by a signal from the GSE. Normally the AGC time counter is stopped at time 12 before and after the execution of an instruction.

Table 32-5
CONTROL PULSES

Pulse	Purpose														
A2X	Copies bits 16 through 1 of register A directly (not through WA's) into bit positions 16 through 1 of register X.														
B15X	Enters a ONE into bit position 15 of register X.														
CI	Inserts carry bit into bit position 1 of the Adder. This adds the quantity one to the content of the Adder if no bit is carried around (from bit positions 16 to bit position 1).														
CLXC	Clears register X if flip-flop BR1 contains a ZERO. (Used in instruction DV E.)														
DVST	Modifies the content of the stage counter (ST) by complementing the content of the next higher bit position as shown below:														
	<table style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Binary</th> <th>Octal</th> </tr> </thead> <tbody> <tr> <td>000</td> <td>0</td> </tr> <tr> <td>001</td> <td>1</td> </tr> <tr> <td>011</td> <td>3</td> </tr> <tr> <td>111</td> <td>7</td> </tr> <tr> <td>110</td> <td>6</td> </tr> <tr> <td>100</td> <td>4</td> </tr> </tbody> </table>	Binary	Octal	000	0	001	1	011	3	111	7	110	6	100	4
Binary	Octal														
000	0														
001	1														
011	3														
111	7														
110	6														
100	4														
EXT	Enters a ONE into bit position EXT of register SQ.														
G2LS 	Copies bits 16, 15 through 4, and 1 of register G directly (not through WA's) into bit positions 16, 12 through 1, and 15 of register X.														
KRPT	Resets interrupt priority cell.														
L16	Enters a ONE into bit position 16 of register L.														
L2GD	Copies bits 16 and 14 through 1 of register L directly (not through WA's) into bit positions 16 and 15 through 2 of register G; enters a ONE into bit position 1 of register G if pulse MCRO is generated.														
MONEX	Clears register X and enters ONE's into bit positions 16 through 2.														
MOU	Causes the generation of one minus drive pulse.														

Table 32-5
CONTROL PULSES (cont)

Pulse	Purpose
NEACOF	Permits end around carry upon completion of subinstruction MP3.
NEACON	Inhibits end around carry (also during WYD) until NEACOF.
NISQ	Causes loading of next instruction into register SQ (implies RB and WSQ at time 12). Also resets the stage counter (ST) to 0; frees certain restrictions; permits execution of instruction RUPT and of all Counter Instructions.
PIFL	Prevents writing into bit position 1 of register Y on control pulse WYD if bit position 15 of register L contains a ONE. (Used in instruction DV.)
PONEX	Clears register X and enters a ONE into bit position 1.
POUT	Causes the generation of one plus drive pulse.
PTWOX	Clears register X and enters a ONE into bit position 2.
R15	Enters 000015 into WA's.
RIC	Enters 177776 (minus one) into WA's.
R6	Enters 000006 into WA's.
RA	Reads bits 16 through 1 of register A into WA's 16 through 1.
RAD	Reads address of next instruction. RAD appears at last time 8 of an instruction and is normally interpreted as RG. If the next instruction is INHINT, RELINT, or EXTEND, RAD is interpreted as RZ and ST2 instead.
RB	Reads bits 16 through 1 of register B into WA's 16 through 1.
RB1	Enters 000001 into WA's.
RB1F	Enters 000001 into WA's if flip-flop BR1 contains a ONE.
RB2	Enters 000002 into WA's.
RBBK	Reads the BB (both bank) configuration into the WA's, i. e., copies the content of bit position 16 of register FBANK into WA's 16 and 15, the content of bit positions 14 through 11 of register FBANK into WA's 14 through 11, and the content of bit positions 11 through 9 of register EBANK into WA's 3 through 1.

Table 32-5
CONTROL PULSES (cont)

Pulse	Purpose
RC	Reads the complemented content of register B (bits 16 through 1 of C) into WA's 16 through 1.
RCH	Reads the content of the input-output channel specified by the contents of register S; bit 15 is read into WA's 16 and 15, and bits 14 through 1 are read into WA's 14 through 1.
RG	Reads bits 16 through 1 of register G into WA's 16 through 1.
RL	Reads bit 16 of register L into WA's 16 and 15, and bits 14 through 1 into WA's 14 through 1.
RL10BB	Reads low 10 bits, i. e., bits 10 through 1 of register B into WA's 10 through 1; replaces c(S), which includes a quarter code, by a 10 bit address.
RQ	Reads bits 16 through 1 of register Q into WA's 16 through 1.
RRPA	Enters into the WA's the address of a RUP T Transfer Routine supplied by the Interrupt Priority Control.
RSC	Reads the content of the CP register specified by the content of register S; bits 16 through 1 are read into WA's 16 through 1.
RSCT	Enters into the WA's the address of a counter address supplied by the Counter Priority Control (paragraph 30-94).
RSTRT	Enters 004000 (Block II start address) into WA's.
RSTSTG	Resets the stage counter to 0 (refer to DVST).
RU	Reads bits 16 through 1 of Adder output gates (U) into WA's 16 through 1.
RUS	Reads bit 15 of Adder output gates (U) into WA's 16 and 15, and bits 14 through 1 into WA's 14 through 1.
RZ	Reads bits 16 through 1 of register Z into WA's 16 through 1.
ST1	Sets stage 1 flip-flop to ONE at next time 12.
ST2	Sets stage 2 flip-flop to ONE at next time 12.
STAGE	Causes the execution of next subinstruction as defined by the content of the stage counter (ST).

Table 32-5
CONTROL PULSES (cont)

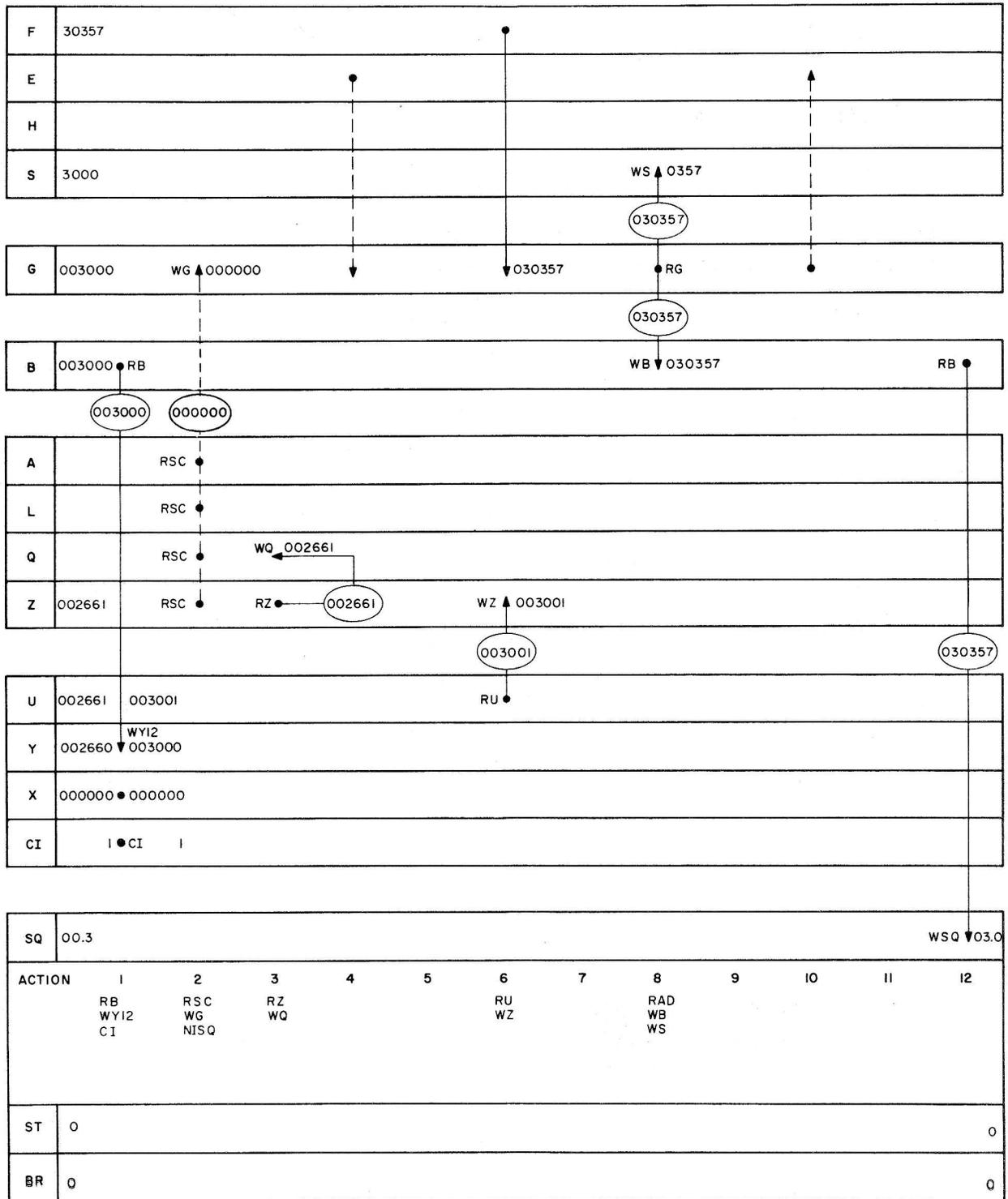
Pulse	Purpose
TL15	Copies bit 15 of register L into flip-flop BR1.
TMZ	Tests the content of the WA's for minus zero: if bits 16 through 1 are all ONE's, flip-flop BR2 is set to ONE; otherwise BR2 is set to ZERO.
TOV	Tests the content of WA's 16 and 15 for overflow: set flip-flops BR1 and BR2 to 01 in case of positive overflow, or to 10 in case of negative overflow.
TPZG	Tests the content of register G for plus zero: if bits 16 through 1 are all ZERO's, flip-flop BR2 is set to ONE; otherwise the content of BR2 is not changed.
TRSM	Tests signals XT1/ and XB7 of selection logic for the resume address (0017) during the execution of subinstruction NDX0: if 0017 is present, subinstruction RSM3 is executed next by setting c(ST) = 3; otherwise subinstruction NDX1 by setting c(ST) = 1.
TSGN	Tests content of WA 16 for sign: if a ZERO, flip-flop BR1 is set to ZERO; if a ONE, flip-flop BR1 is set to ONE without changing the content of flip-flop BR2.
TSGN2	Tests content of WA 16 for sign: if a ZERO, flip-flop BR2 is set to ZERO; if a ONE, flip-flop BR2 is set to ONE without changing the content of flip-flop BR1.
TSGU	Tests content of output gate U16 of Adder for sign: if a ZERO, flip-flop BR1 is set to ZERO; if a ONE, flip-flop BR1 is set to ONE.
U2BBK	Copies bits 16 and 14 through 11 of the Adder output gates (U) into bit positions 16 and 14 through 11 of register FBANK, and bits 3 through 1 (of U) into bit positions 3 through 1 of register EBANK. U2BBK may be inhibited by signal MONWBK, which is generated if register BBANK is addressed.
WA	Clears register A and writes the content of WA's 16 through 1 into bit positions 16 through 1.

Table 32-5
CONTROL PULSES (cont)

Pulse	Purpose
WALS 	Clears register A and writes the content of WA's 16 through 3 into bit positions 14 through 1. If bit position 1 of register G contains a ZERO, the content of bit position 16 of register G is entered into bit positions 16 and 15 of register A; if bit position 1 of register G contains a ONE, the content of output gate U 16 of the Adder is entered into bit positions 16 and 15 of register A. WALS also clears bit positions 14 and 13 of register L, and writes the content of WA's 2 and 1 into these bit positions.
WB	Clears register B and writes the content of WA's 16 through 1 into bit positions 16 through 1.
WCH	Clears the output channel specified by the content of register S and writes the content of WA's 16 and 14 through 1 into bit positions 15 through 1.
WG	Clears register G and writes the content of WA's 16 through 1 into bit positions 16 through 1, except if register S contains addresses 0020 through 0023 in which case the WA content is cycled or shifted (paragraph 30-41).
WL	Clears register L and writes the content of WA's 16 through 1 into bit positions 16 through 1.
WOVR	Tests the content of WA's 16 and 15 for positive overflow: if register S contains 0025, counter 0024 is incremented; if register S contains 0026, 0027, or 0030, instruction RUPT is executed.
WQ	Clears register Q and writes the content of WA's 16 through 1 into bit positions 16 through 1.
WS	Clears register S and writes the content of WA's 12 through 1 into bit positions 12 through 1.
WSC	Clear the CP register specified by the content of register S and writes the content of WA's 16 through 1 into bit positions 16 through 1.
WSQ 	Clears register SQ and writes the content of WA's 16 and 14 through 10 into bit positions 16 and 14 through 10.

Table 32-5
CONTROL PULSES (cont)

Pulse	Purpose																																																															
WY	Clears registers X and Y and carry flip-flop CI; writes the content of WA's 16 through 1 into bit positions 16 through 1 of register Y.																																																															
WY12	Clears registers X and Y and carry flip-flop CI; writes the content of WA's 12 through 1 into bit positions 12 through 1 of register Y.																																																															
WYD	Clears registers X and Y and carry flip-flop CI, writes the content of WA's 16 and 14 through 1 into bit positions 16 and 15 through 2 of register Y; writes the content of WA 16 into bit position 1 of register Y except in SHINC sequence, or unless bit position 15 of register L contains a ONE at PIFL, or if end around carry is inhibited by control pulse NEACON.																																																															
WZ	Clears register Z and writes the content of WA's 16 through 1 into bit positions 16 through 1.																																																															
Z15	Enters a ONE into bit position 15 of register Z.																																																															
Z16	Enters a ONE into bit position 16 of register Z.																																																															
ZAP	Causes the generation of control pulses RU, G2LS, and WALS (used in instruction MP K).																																																															
ZIP	Causes generation of control pulses A2X and L2GD (used in instruction MP K); performs read/write operations depending on the content of bit positions 15, 2, and 1 of register L as shown: <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>L15</th> <th>L2</th> <th>L1</th> <th>Read</th> <th>Write</th> <th>Carry</th> <th>Remember</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>-</td> <td>WY</td> <td>-</td> <td>-</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>RB</td> <td>WY</td> <td>-</td> <td>-</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>RB</td> <td>WYD</td> <td>-</td> <td>-</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>RC</td> <td>WY</td> <td>CI</td> <td>MCRO</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>RB</td> <td>WY</td> <td>-</td> <td>-</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>RB</td> <td>WYD</td> <td>-</td> <td>-</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>RC</td> <td>WY</td> <td>CI</td> <td>MCRO</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>-</td> <td>WY</td> <td>-</td> <td>MCRO</td> </tr> </tbody> </table> <p style="margin-left: 20px;">If MCRO occurs, a ONE is entered into L15</p>	L15	L2	L1	Read	Write	Carry	Remember	0	0	0	-	WY	-	-	0	0	1	RB	WY	-	-	0	1	0	RB	WYD	-	-	0	1	1	RC	WY	CI	MCRO	1	0	0	RB	WY	-	-	1	0	1	RB	WYD	-	-	1	1	0	RC	WY	CI	MCRO	1	1	1	-	WY	-	MCRO
L15	L2	L1	Read	Write	Carry	Remember																																																										
0	0	0	-	WY	-	-																																																										
0	0	1	RB	WY	-	-																																																										
0	1	0	RB	WYD	-	-																																																										
0	1	1	RC	WY	CI	MCRO																																																										
1	0	0	RB	WY	-	-																																																										
1	0	1	RB	WYD	-	-																																																										
1	1	0	RC	WY	CI	MCRO																																																										
1	1	1	-	WY	-	MCRO																																																										
ZOUT	Stops the generation of drive pulses.																																																															
	This pulse does not appear in the pulse sequences; refer to ZAP.																																																															
	This pulse does not appear in the pulse sequences; refer to NISQ.																																																															



2701A

Figure 32-2. Subinstruction TC0

- (1) Retain $c(Q)$.
- (2) Set $c(B) = c(F) = f$, f being the instruction stored at location F .
Set $c(S) =$ relevant address of f .
Set $c(SQ) =$ order code of f .
- (3) Set $c(Z) = F + 1$.

Point (2) implies that instruction f is executed next.

32-41. There are no restrictions on instruction TCF F , or special cases, except that F must represent an address in F Memory.

32-42. The execution of subinstruction TCF0 is similar to that of subinstruction TC0 except for action 3 which has no control pulse (row 3 of table 32-2). The content of register Q is not changed. The relevant address contained in register B is incremented by one and entered into register Z .

32-43. INSTRUCTION CCS E

32-44. Instruction CCS E (Count, Compare, and Skip on E) is a Basic Instruction which is represented by order code 01.0 and a 10-bit address. Instruction CCS E consists of subinstructions CCS0 and STD2, the execution of which takes two MCT's.

32-45. Instruction CCS E examines the data stored at location E in E Memory (or in a CP register) and branches accordingly. The operation CCS E with $0024 \leq E \leq 17777$ can be formulated as follows:

If $c(E)$ is positive non-zero, i. e., if $00001 \leq c(E) \leq 37777$:

- (1) Set $c(A) = c(E) - 1$.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction CCS E, and j being the instruction stored at location $(I+1)$.
Set $c(S) =$ relevant address of j .
Set $c(SQ) =$ order code of j .
- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(E) = b(E)$ and $c(I+1) = b(I+1)$ if E and/or $(I+1)$ represent an address in E Memory.

If $c(E)$ is plus zero, i. e., if $c(E) = 00000$:

- (1) Set $c(A) = 000000$
- (2) Set $c(B) = c(I+2) = j$, I being the address of instruction CCS E, and j being the instruction stored at location $(I+2)$.
- (3) Set $c(Z) = b(Z)+2 = I+3$.

- (4) Restore $c(E) = b(E)$ and $c(I+2) = b(I+2)$ if E and/or $(I+2)$ represent an address in E Memory.

If $c(E)$ is negative non-zero, i. e., if $40000 \leq c(E) \leq 77776$:

- (1) Set $c(A) = \bar{c}(E) - 1$, \bar{c} for complemented content.
- (2) Set $c(B) = c(I+3) = j$, I being the address of instruction CCS E , and J being the instruction stored at location $(I+3)$.
Set $c(S) =$ relevant address of j .
Set $c(SQ) =$ order code of j .
- (3) Set $c(Z) = b(Z) + 3 = I+4$
- (4) Restore $c(E) = b(E)$ and $c(I+3) = b(I+3)$ if E and/or $(I+3)$ represent an address in E Memory.

If $c(E)$ is minus zero, i. e., if $c(E) = 17777$:

- (1) Set $c(A) = 000000$.
- (2) Set $c(B) = c(I+4) = j$, I being the address of instruction CCS E , and j being the instruction stored at location $(I+4)$.
Set $c(S) =$ relevant address of j .
Set $c(SQ) =$ order code of j .
- (3) Set $c(Z) = b(Z) + 4 = I+5$.
- (4) Restore $c(E) = b(E)$ and $c(I+4) = b(I+4)$ if E and/or $(I+4)$ represent an address in E Memory.

Point (2) of all cases implies that the respective instruction j is executed next.

32-46. Special Cases of CCS E :

- a. CCS A , a very useful instruction, examines the data in A ; however, the $b(A)$ is changed.
- b. CCS L , CCS Q , and CCS Z follow the rules of paragraph 32-45.
- c. CCS $EBANK$, CCS $FBANK$, and CCS $BBANK$ also follow the rules of paragraph 32-45; however, the particular read and write operations must be observed.
- d. CCS $ZERO$ has no purpose.
- e. Instructions CCS E with $0010 \leq E \leq 0017$ follow the rules of paragraph 32-45.
- f. Instructions CCS E with $0020 \leq E \leq 0023$ also follow the rules of paragraph 32-45 except that $c(E)$ is edited during restoring.

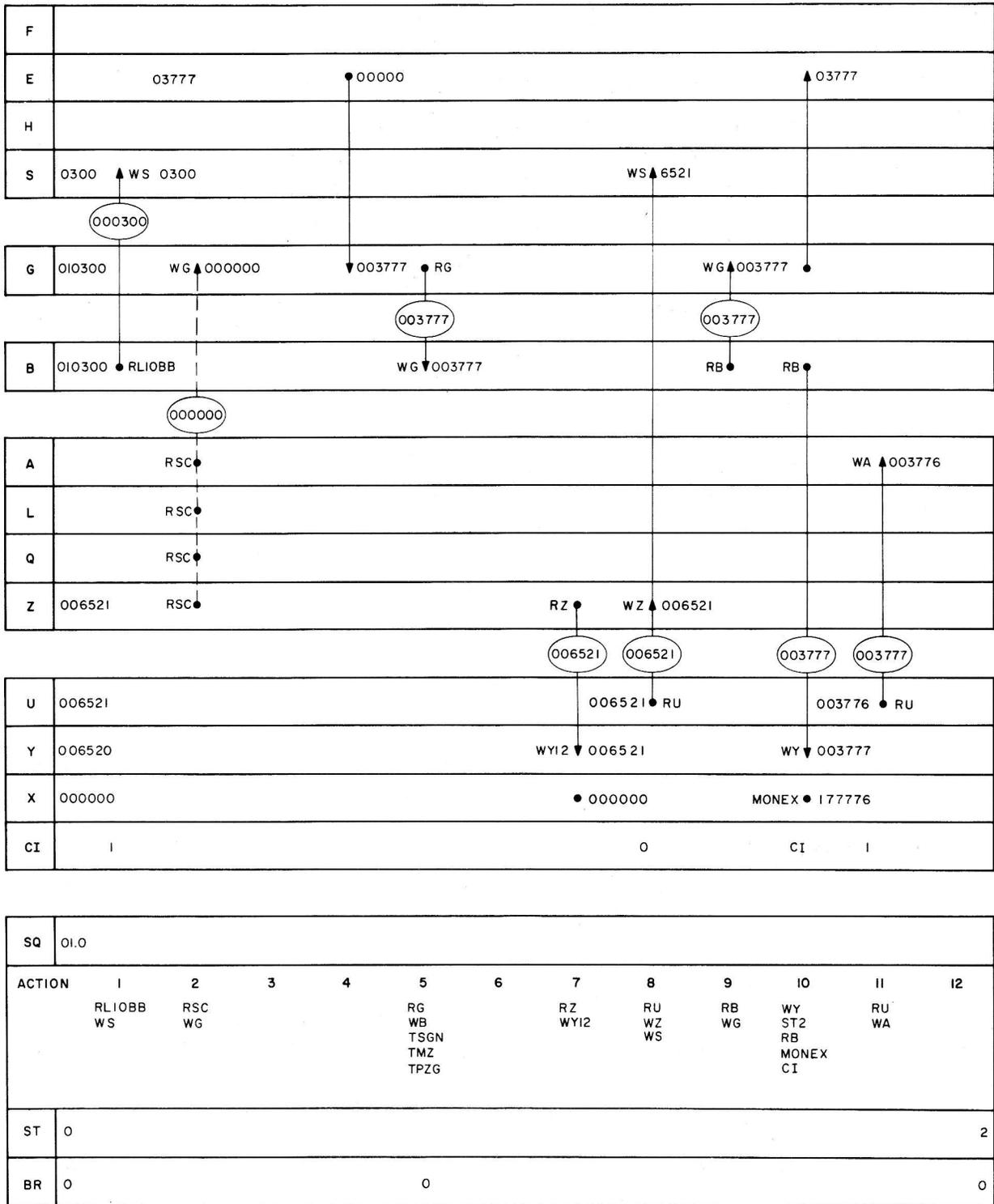
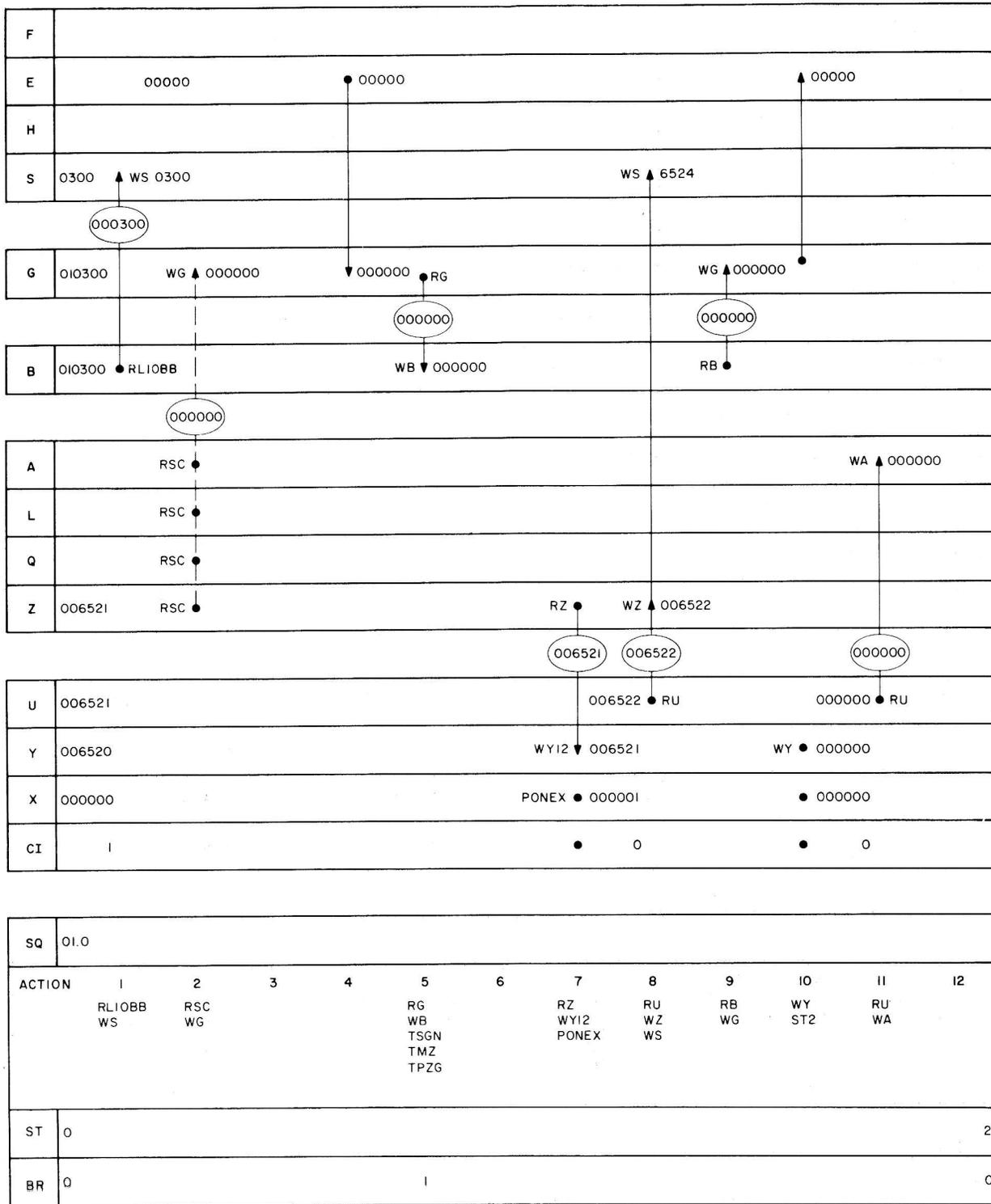
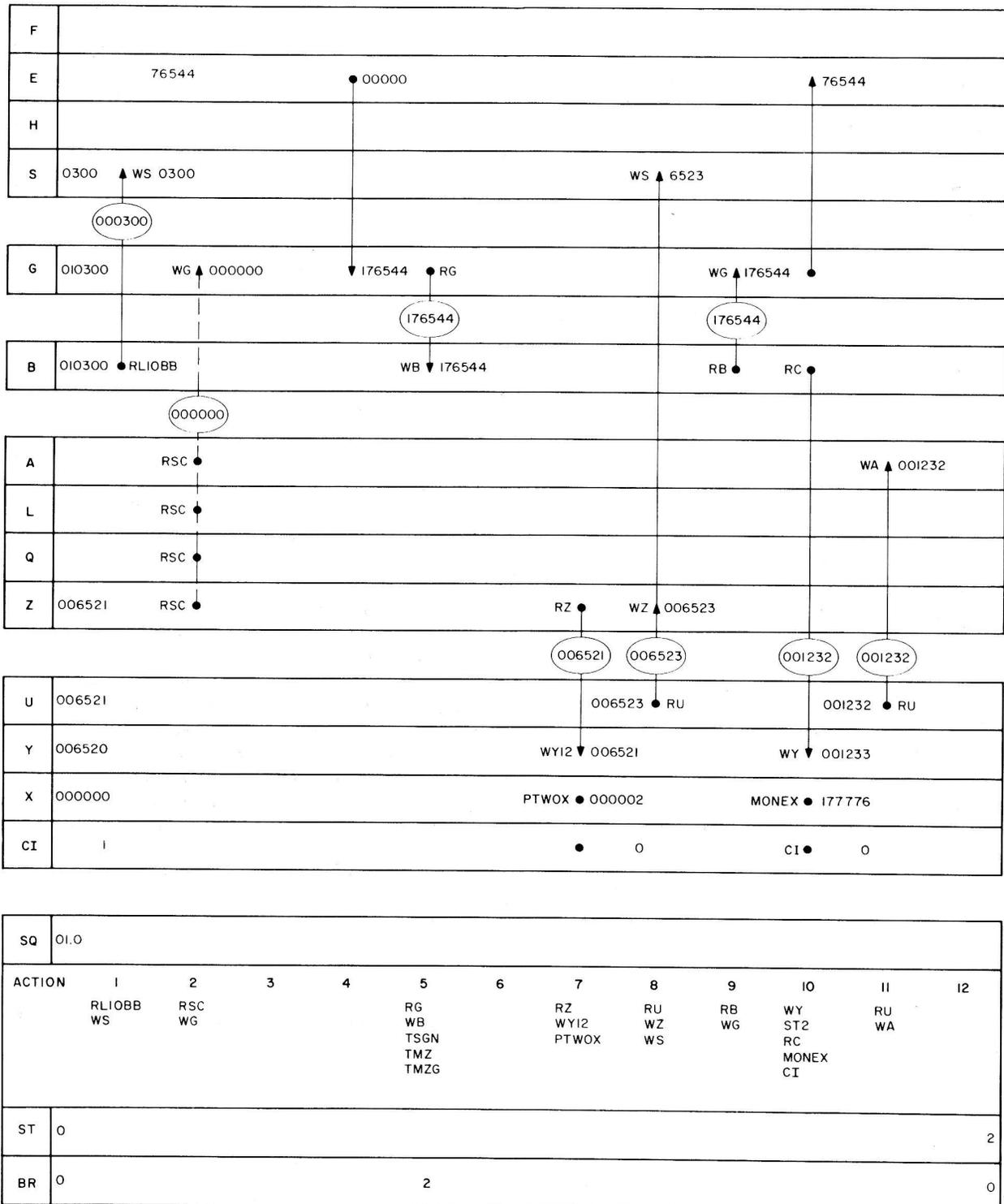


Figure 32-3. Subinstruction CCS0, Branch on Quantity Greater than Plus Zero



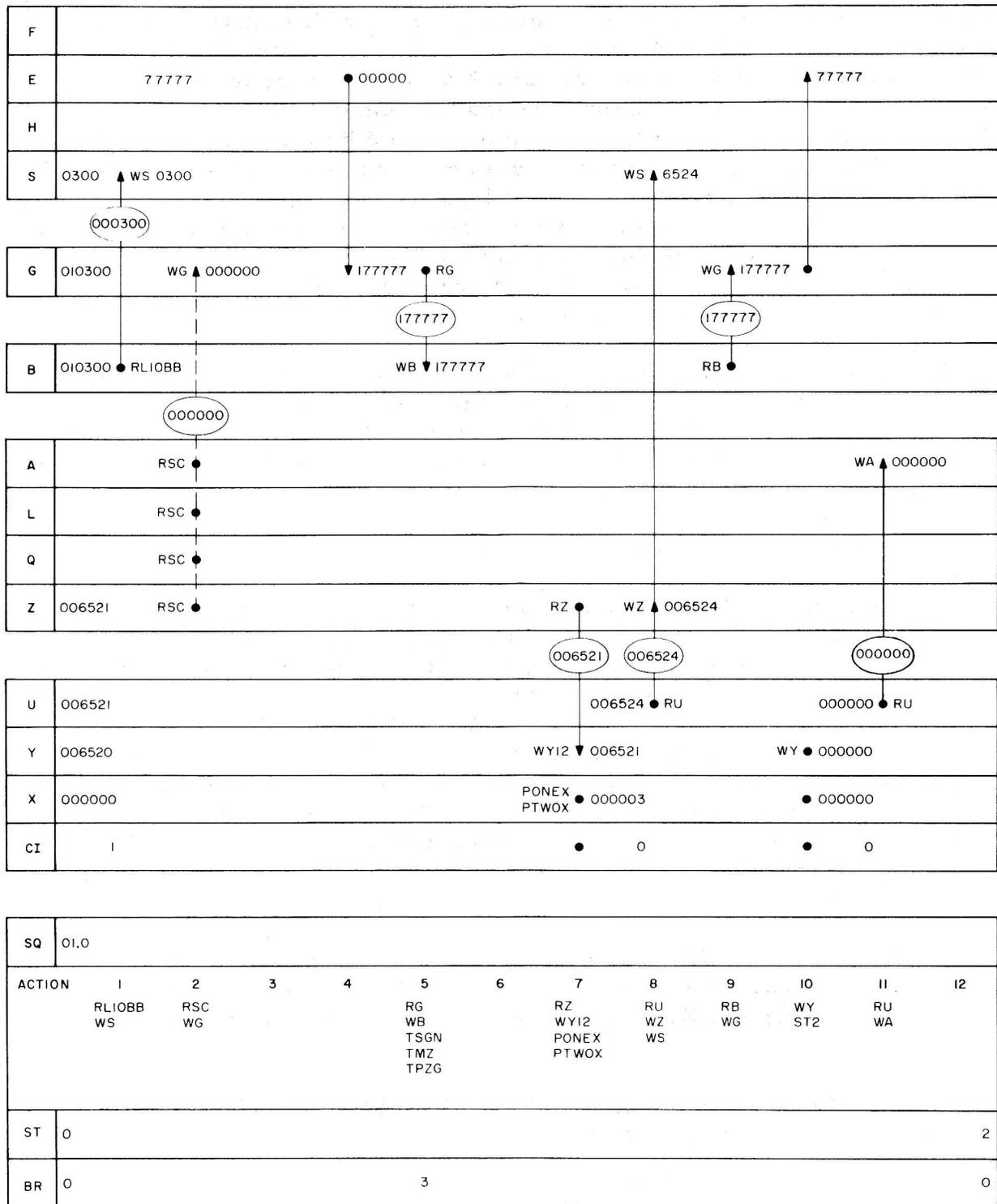
2705A

Figure 32-4. Subinstruction CCS0, Branch on Plus Zero



2704A

Figure 32-5. Subinstruction CCS0, Branch on Quantity Less than Minus Zero



2703A

Figure 32-6. Subinstruction CCS0, Branch on Minus Zero

32-47. When instruction CCS E is executed, action 1 of subinstruction CCS0 (row 4 of table 32-2) enters the relevant address of instruction CCS E into register S. At time 1 of the first subinstruction, register B always contains the instruction to be executed. At time 2 or 4, the quantity to be tested is entered into register G. Action 5 enters the quantity into register B and sets the branch flip-flops accordingly. Actions 7 and 8 increment the content of register Z by 0, 1, 2, or 3 to specify a new "next address" and enter it into register S. Action 9 returns the tested quantity to register G for restoring in memory. Action 10 diminishes the tested non-zero quantity by one and action 11 enters the diminished quantity into register A. Subinstruction STD2 increments the content of register Z by one and calls forward the instruction defined by the previous content of register Z.

32-48. The execution of subinstruction CCS0 of CCS 0300, with location 0300 containing a quantity greater than plus zero, is illustrated in figure 32-3. Figures 32-4, 32-5, and 32-6 illustrate the execution of the same instruction with location 0300 containing different quantities (plus zero, less than minus zero, and minus zero).

32-49. INSTRUCTION BZF F

32-50. Instruction BZF F (Branch on Zero to Fixed F) is an Extra Code Instruction which is represented by order code 11.2, 11.4, or 11.6 and a 12 bit address. The address contains a ONE in bit position 11 or 12 or in both. The order code is composed of 11. plus the two address bits mentioned. Instruction BZF F must be preceded by Special Instruction EXTEND which enters a ONE into bit position EXT of register SQ. (Code 1. is taken from bit positions 16, 14, and 13 of register B, and entered into the corresponding bit positions of register SQ.) Instruction BZF F consists of subinstruction BZF0 only if branching occurs, and of subinstructions BZF0 and STD2 if no branching occurs; consequently, the execution of instruction BZF F may take one or two MCT's.

32-51. Instruction BZF F examines the data contained in register A and takes the next instruction from location F in F memory if register A contains zero. The operation BZF F with $2000 \leq F \leq 7777$ can be formulated as follows:

If $c(A)$ is nonzero, i. e., if $000001 \leq c(A) \leq 077777$ or $100000 \leq c(A) \leq 177776$:

- (1) Retain $c(A)$
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction BZF F, and j being the instruction stored at location (I+1).

- Set $c(S)$ = relevant address of j .
 Set $c(SQ)$ = order code of j .
- (3) Set $c(Z) = b(Z)+1=I+2$
 - (4) Restore $c(I+1) = b(I+1)$ if $(I+1)$ represents an address in E Memory.

Point (2) implies that instruction j is executed next.

If $c(A)$ is zero, i. e., if $c(A) = 000000$ or $c(A) = 177777$:

- (1) Retain $c(A)$
- (2) Set $c(B) = c(F) = f$, f being the instruction stored at location F .
 Set $c(S)$ = relevant address of f .
 Set $c(SQ)$ = relevant address of f .
- (3) Set $c(Z) = F+1$.

Point (2) implies that instruction f is executed next.

32-52. There are no restrictions on instruction BZF F, or special cases, except that F must represent an address in F Memory.

32-53. When instruction BZF F is executed, action 1 of subinstruction BZF0 (row 5 of table 32-2) enters the content of register A into register G. Actions 1 and 2 set the branch flip-flops (BR) to 0 or 2 if a non-zero quantity has been entered, or to 1 or 3 if the quantity zero has been entered into register G. If a non-zero quantity has been entered, the address of the next instruction is copied from register Z into register S by action 8 and subinstruction STD2 increments the content of register Z and calls forward the next instruction. If registers A and G contain 000000 or 177777 at time 2, the relevant address F of instruction BZF F contained in register B is incremented by one by action 5 and entered into register Z by action 6. (Compare with actions 1 and 6 of subinstruction TC0.) At time 6, the instruction stored at location F is entered into register G and copied into register B by action 8. Action 8 also enters the relevant address of the instruction into register S and enters its order code into register SQ at time 12.

32-54. The execution of subinstruction BZF0 of BZF 6055, with A containing 004765, and location 6055 containing instruction CA 0221 (30221), is illustrated in figure 32-7. Figure 32-8 illustrates the execution of the same instruction when A contains 000000. When A contains any negative quantity such as 176543, BR is set to 2 by control pulse TSGN. Otherwise, figure 32-7 applies. When A contains 177777, BR is set to 3 by control pulses TSGN and TMZ; otherwise figure 32-8 applies.

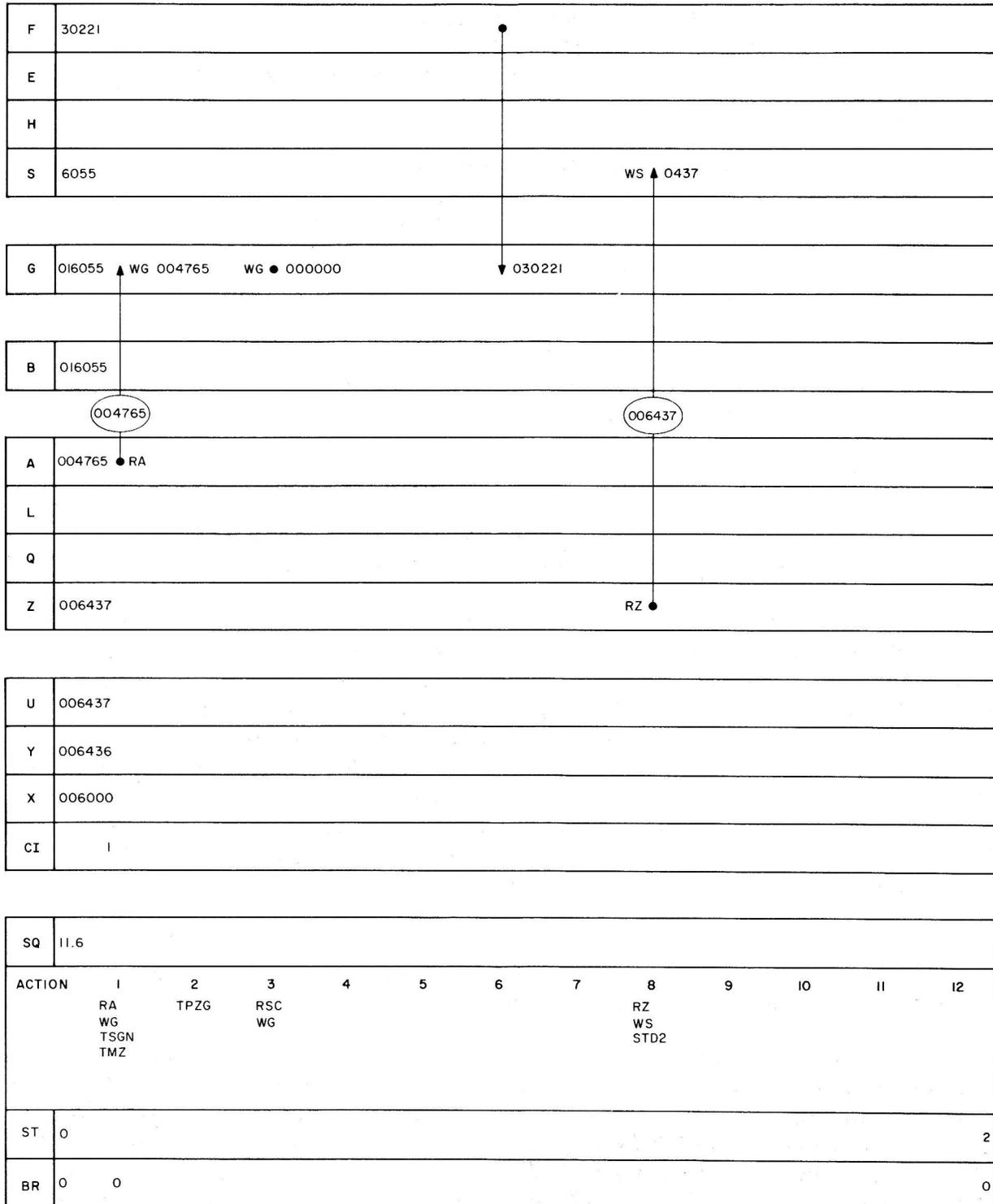
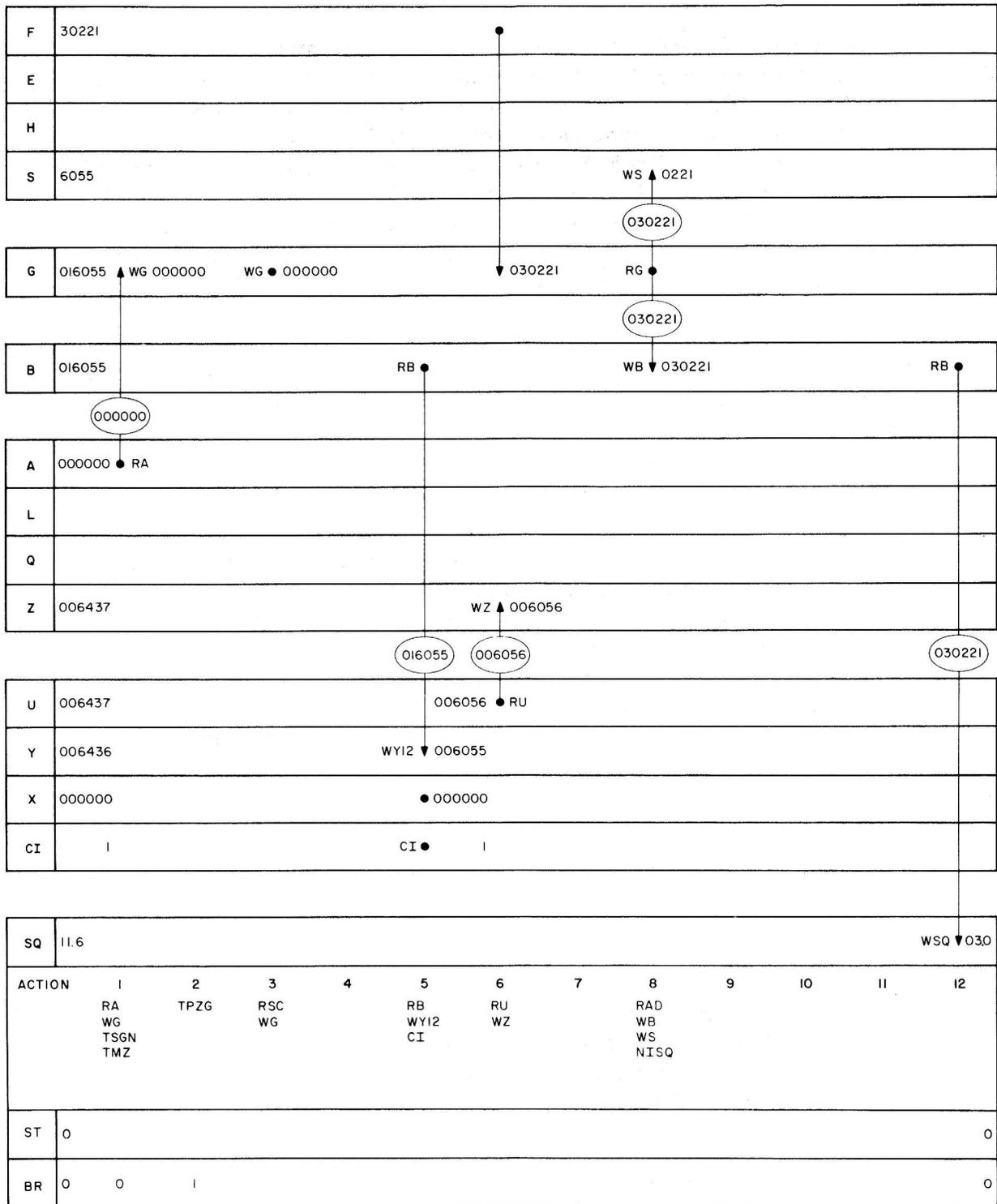


Figure 32-7. Subinstruction BZF0, With Register A Containing a Positive Non-Zero Quantity



2786A

Figure 32-8. Subinstruction BZF0, With Register A containing Plus Zero

32-55. INSTRUCTION BZMF F

32-56. Instruction BZMF F (Branch on Zero or Minus to Fixed F) is an Extra Code Instruction which is represented by order code 16.2, 16.4, or 16.6 and a 12 bit address. The address contains a ONE in bit position 11 or 12, or in both. The order code is composed of 16, plus the two address bits mentioned. Instruction BZMF F must be preceded by Special Instruction EXTEND which enters a ONE into bit position EXT of register SQ. Instruction BZMF F consists of subinstruction BZMF0 only if branching occurs, and of subinstruction BZMF0 and STD2 if no branching occurs; consequently, the execution of instruction BZMF F may take one or two MCT's.

32-57. Instruction BZMF F examines the data contained in register A and takes the next instruction from location F in F Memory if register A contains zero or a negative non-zero quantity. The operation BZMF F with $2000 \leq F \leq 7777$ can be formulated as follows:

If $c(A)$ is positive non-zero, i. e., if $000001 \leq c(A) \leq 077777$:

- (1) Retain $c(A)$.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction BZF F, and j being the instruction stored at location (I+1).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(I+1) = b(I+1)$ if (I+1) represents an address in E Memory.

Point (2) implies that instruction j is executed next.

If $c(A)$ is not positive non-zero, i. e., if $c(A) = 000000$ or $100000 \leq c(A) \leq 177777$:

- (1) Retain $c(A)$
- (2) Set $c(B) = c(F) = f$, f being the instruction stored at location F.
Set $c(S) =$ relevant address of f.
Set $c(SQ) =$ relevant address of f.
- (3) Set $c(A) = F+1$.

Point (2) implies that instruction f is executed next.

32-58. There are no restrictions on instruction BZMF F, or special cases, except that F must represent an address in F Memory.

32-59. The execution of instruction BZMF F is similar to that of instruction BZF F. Both subinstructions BZF0 and BZMF0 set the branch flip-flops by actions 1 and 2; however, depending on the content of the branch flip-flops, actions 5, 6, and 7 operate differently. Refer to rows 5 and 6 of table 32-4. When A contains a positive non-zero quantity, subinstruction BZMF0 sets BR to 0 and no branching occurs. When A contains plus zero, a negative non-zero quantity, or minus zero, BR is set to 1, 2, or 3, respectively, and branching occurs.

32-60. FETCHING AND STORING INSTRUCTIONS

32-61. INSTRUCTION CA K

32-62. Instruction CA K (Clear and Add K) is a Basic Instruction which is represented by order code 03, and a 12 bit address. Instruction CA K consists of subinstructions CA0 and STD2, the execution of which takes two MCT's. Alternate spelling CAF K is permitted when K refers to a location in F Memory, and CAE K when K refers to a location in E Memory or a CP register.

32-63. Instruction CA K clears register A and enters into it the data stored at location K. The operation CA K with $0024 \leq K \leq 7777$ can be formulated as follows:

- (1) Set $c(A) = c(K)$
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction CA K, and j being the instruction stored at location (I+1).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(K) = b(K)$ and $c(I+1) = b(I+1)$ if K and/or (I+1) represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-64. Special Cases of CA K:

- a. CA A (alternate code NOOP, for no operation) has no effect.
- b. CA L, CA Q, and CA Z follow the rules of paragraph 32-63.
- c. CA EBANK, CA FBANK, and CA BBANK can be used; however, the particular read and write operations have to be observed.
- d. CA ZERO enters 000000 into A.

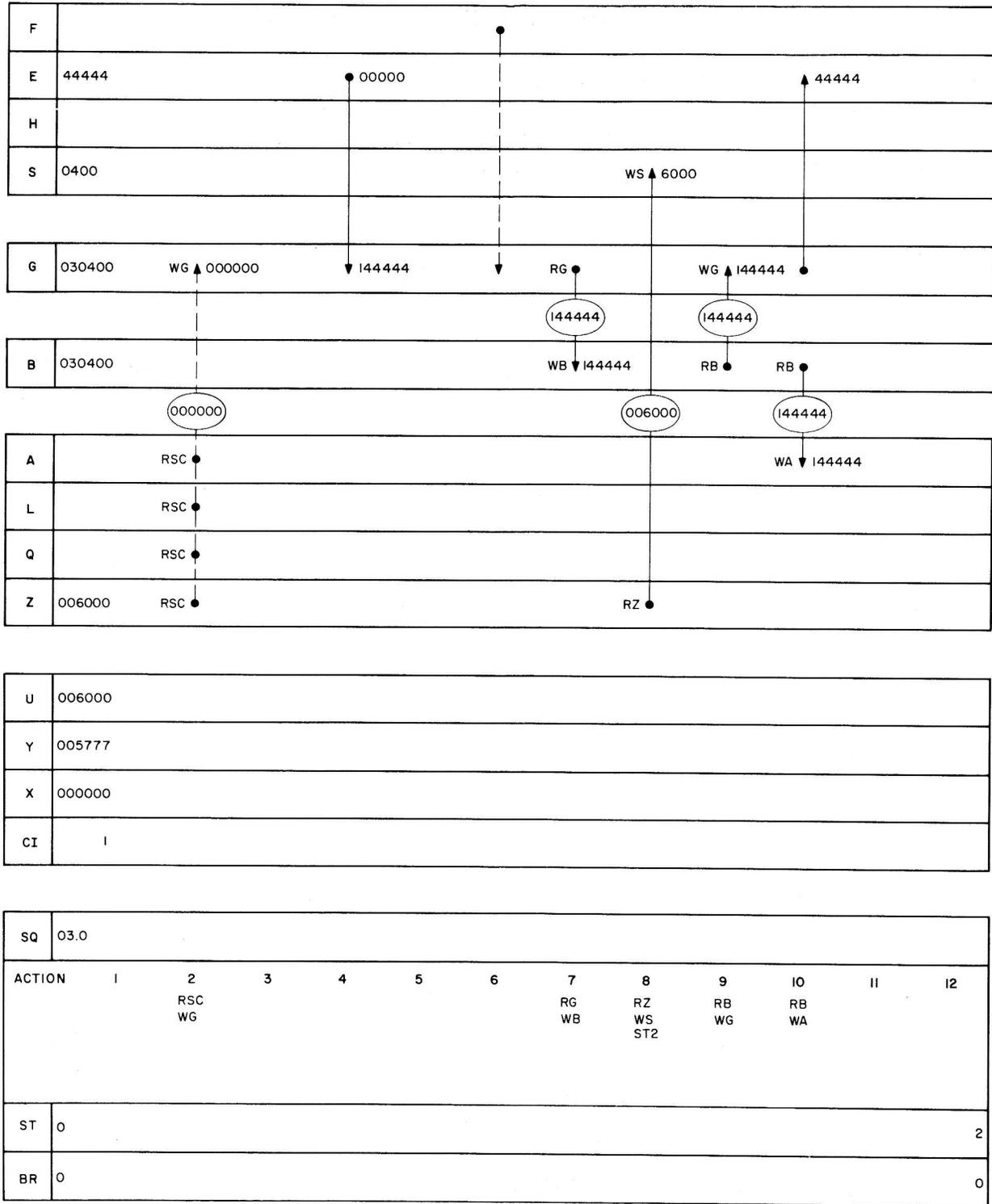


Figure 32-9. Subinstruction CA0

- e. Instructions CA K with $0010 \leq K \leq 0017$ follow the rules of paragraph 32-63.
- f. Instructions CA K with $0020 \leq K \leq 0023$ also follow the rules of paragraph 32-63, except that the $c(K)$ is edited during restoring.

32-65. When instruction CA K is executed, action 2 of subinstruction CA0 (row 7 of table 32-4) enters the desired quantity into register G if this quantity is stored in a CP register. Otherwise, register G is cleared and the desired quantity is entered by E Memory at time 4, or by F Memory at time 6. Action 7 copies this quantity into register B, and action 10 copies the quantity (from register B) into register A. Action 9 returns the quantity to register G for restoring in E Memory. Subinstruction STD2 increments the content of register Z by one and calls forward the instruction defined by the previous content of register Z.

32-66. The execution of subinstruction CA0 of CA0400, with location 0400 containing 44444, is illustrated in figure 32-9.

32-67. INSTRUCTION CS K

32-68. Instruction CA K (Clear and Subtract K) is a Basic Instruction which is represented by order code 04. and a 12 bit address. Instruction CS K consists of subinstructions CS0 and STD2, the execution of which takes two MCT's.

32-69. Instruction CS K clears register A and enters into it the complemented data stored at location K. The operation CS K with $0024 \leq K \leq 7777$ can be formulated as follows:

- (1) Set $c(A) = \bar{c}(K)$, \bar{c} for complemented content.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction CS K, and j being the instruction stored at location $E(I+1)$.
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(K) = b(K)$ and $c(I+1) = b(I+1)$ if K and/or (I+1) represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-70. Special Cases of CS K:

- a. CS A (alternate code COM, for complement) complements the content of A.
- b. CS L, CA Q, and CA Z follow the rules of paragraph 32-70.
- c. CS EBANK, CA FBANK, and CA BBANK can be used; however, the particular read and write operations must be observed.
- d. CS ZERO enters 177777 into A.
- e. Instructions CS K with $0010 \leq K \leq 0017$ follow the rules of paragraph 32-69.
- f. Instructions CS K with $0020 \leq K \leq 0023$ also follow the rules of paragraph 32-69, except that the $c(K)$ is edited during restoring.

32-71. The execution of instruction CS K is similar to that of instruction CA K (compare instructions CA 0 and CS 0 in rows 7 and 8 of table 32-4). Action 10 of CA0 takes the desired quantity from the normal read side of register B and enters this quantity into register A. Action 10 of CS0 takes the desired quantity from the complement read side of register B and enters the complemented quantity into register A.

32-72. INSTRUCTION DCA K

32-73. Instruction DCA K (Double Clear and Add K) is an Extra Code Instruction which is represented by order code 13. and a 12 bit address. Instruction DCA K must be preceded by Special Instruction EXTEND which enters a ONE into bit position EXT of register SQ. (Code 3. is taken from bit positions 16, 14, and 13 of register B and entered into the corresponding bit positions of register SQ.) Instruction DCA K consists of subinstructions DCA0, DCA1, and STD2, the execution of which takes three MCT's.

32-74. Instruction DCA K clears registers A and L and enters into them the data stored at locations K and K+1, respectively. The operation DCA K with $0024 \leq K \leq 7776$, excluding the last address of any E or F Memory bank, (tables 30-3 and 30-4) can be formulated as follows:

- (1) Set $c(A) = c(K)$
Set $c(L) = c(K+1)$
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction DCA K and j being the instruction stored at location (I+1).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z)+1 = I+2$.

- (4) Restore $c(K) = b(K)$, $c(K+1) = b(K+1)$, and $c(I+1) = b(I+1)$ if K , $(K+1)$ and/or $(I+1)$ represent an address in E Memory.

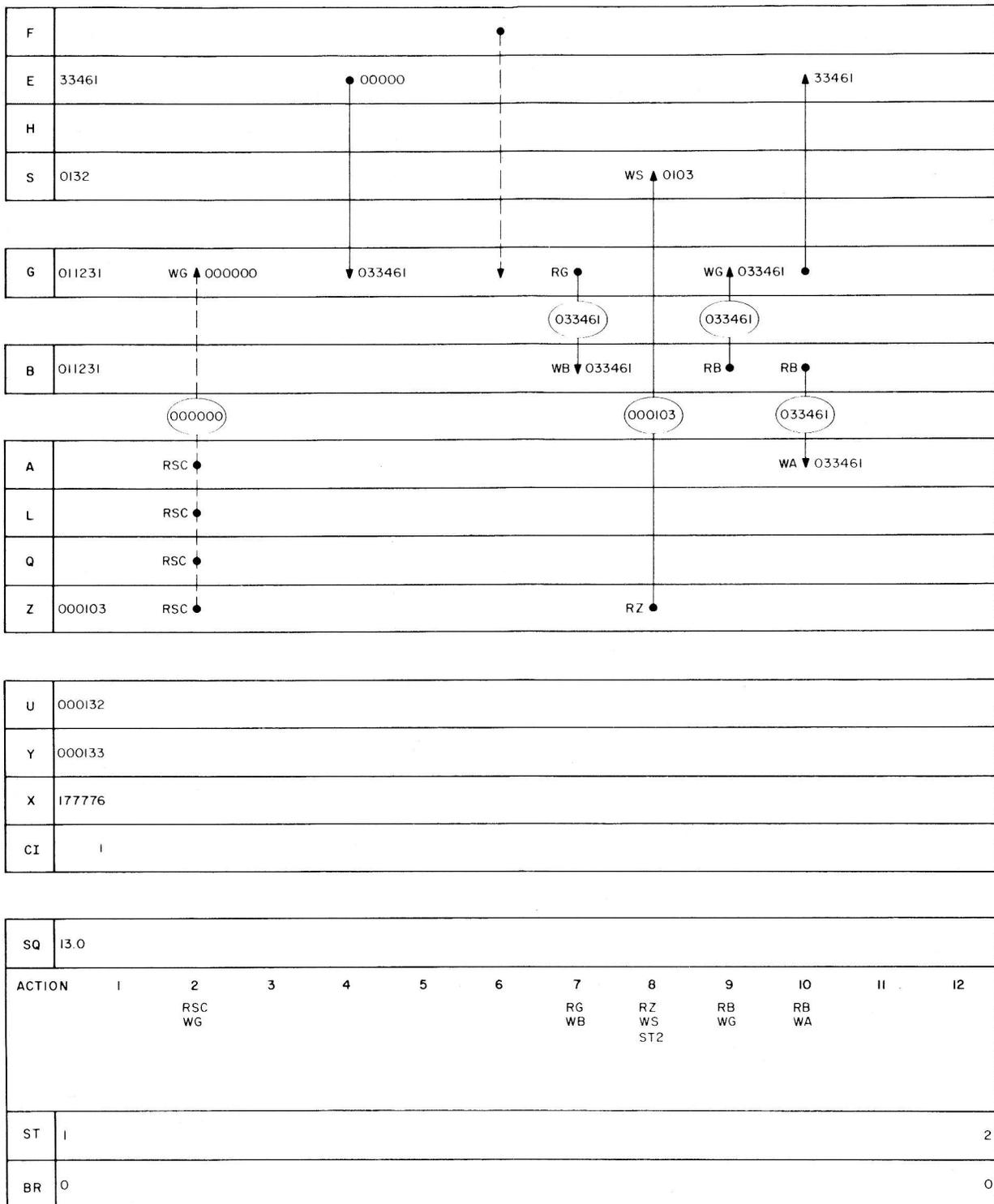
Point (2) implies that instruction j is executed next.

32-75. Special cases of DCA K :

- a. DCA A has no purpose.
- b. DCA 0010 (DCA ARUPT) and DCA 0013 (table 30-4) are useful and follow the rules of paragraph 32-74.
- c. Any DCA K with $0000 \leq K \leq 0022$ must be used with extreme care so as not to destroy stored data; K must not be 0023 in order to prevent destruction of data in counter T2. Whenever locations 0020 through 0023 are involved, the content is edited during the restoring operation.

32-76. When instruction DCA K is executed, first the content of location $(K+1)$ is entered into register L by subinstruction DCA0, then the content of location K is entered into register A by subinstruction DCA1. The Yul Programming System accomplishes this by replacing instruction DCA K with code DCA $(K+1)$ which is wired into the program. As the AGC executes subinstruction DCA0, relevant address $(K+1)$ is available first and decremented by one. Subinstruction DCA1 then uses the decremented address K . For execution of subinstructions DCA0 and DCA1, refer to rows 9 and 10 of table 32-4. When double precision quantities are taken from memory, K must not be equal to the last address of any E or F memory bank in order to allow $K+1$ to be a legal address.

32-77. The execution of instruction DCA 0132 is illustrated in figures 32-10 and 32-11. It is assumed that this instruction is stored at location 2103. Location 0132 contains quantity 33461 and location 0133 contains quantity 11231. Note that registers B, G, and S contain relevant address 0133 at the start of subinstruction DCA0 instead of 0132. Thus, E Memory enters the quantity 11231 into register G at time 4 and actions 7 and 10 bring the quantity into register L. Action 1 decrements address 0133 and action 8 enters the decremented address 0132 into register S. At time 4 of subinstruction DCA1, E Memory enters the quantity 33461 into register G and actions 7 and 10 bring this quantity into register A. Action 8 enters the address of the next instruction stored in register Z into register S as usual, and subinstruction STD2 increments the content of register Z and calls forward the next instruction.



2778A

Figure 32-11. Subinstruction DCA1

32-78. INSTRUCTION DCS K

32-79. Instruction DCS K (Double Clear and Subtract K) is an Extra Code Instruction which is represented by order code 14. and a 12 bit address. Instruction DCS K must be preceded by special instruction EXTEND which enters a ONE into bit position EXT of register SQ. Instruction DCS K consists of subinstructions DCS0, DCS1, and STD2, the execution of which takes three MCT's.

32-80. Instruction DCS K clears registers A and L, complements the data stored at locations K and K+1, and enters this data into registers A and L, respectively. The operation DCS K with $0024 \leq K \leq 7776$, excluding the last address of any E or F memory bank (tables 30-3 and 30-4), can be formulated as follows:

- (1) Set $c(A) = \overline{c(K)}$
Set $c(L) = c(K+1)$
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction DCA K and j being the instruction stored at location (I+1).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z) + 1 = I+2$.
- (4) Restore $c(K) = b(K)$, $c(K+1) = b(K+1)$ and $c(I+1) = b(I+1)$ if K, (K+1), and/or (I+1) represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-81. Special cases of DCS K:

- a. DCS A, (alternate code DCOM, for double precision complement) complements the contents of registers A and L.
- b. DCS 0010 and DCS 0013 (table 30-4) may be useful and follow the rules of paragraph 32-80.
- c. Any other DCS K with $0000 \leq K \leq 0022$ must be used with extreme care so as not to destroy stored data; K must not be 0023 in order to prevent destruction of data in counter T2. Whenever locations 0020 through 0023 are involved, the content is edited during the restoring operation.

32-82. The execution of instruction DCS K is similar to that of instruction DCA K. Compare rows 11 and 12 of table 32-4 with rows 9 and 10.

In action 10 of subinstructions DCS0 and DCS1, the control pulse RB is replaced by pulse RC to read the complemented quantity instead of the non-complemented quantity out of register B.

32-83. INSTRUCTION TS E

32-84. Instruction TS E (Transfer to Storage E) is a Basic Instruction which is represented by order code 05.4 and a 10 bit address. Instruction TS E consists of subinstructions TS0 and STD2, the execution of which takes two MCT's.

32-85. Instruction TS E enters the content of register A into location E of E Memory (or a CP register) and skips if A contains an overflow bit. The operation TS E with $0024 \leq E \leq 17777$ can be formulated as follows:

If register A does not contain an overflow bit:

- (1) Set $c(E) = c(A)$.
Retain $c(A)$.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction TS E, and j being the instruction stored at location (I+1).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(I+1) = b(I+1)$ if (I+1) represents an address in E Memory.

If register A contains an overflow bit:

- (1) Set $c(E) = c(A)$ except for overflow bit.
Set $c(A) = 000001$ if A contained a positive overflow.
Set $c(A) = 177776$ if A contained a negative overflow.
- (2) Set $c(B) = c(I+2) = j$, I being the address of instruction TS E, and j being the instruction located at (I+2).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z)+2 = I+3$.
- (4) Restore $c(I+2) = b(I+2)$ if (I+2) represents an address in E Memory.

Point (2) of both cases implies that the respective instruction j is executed next.

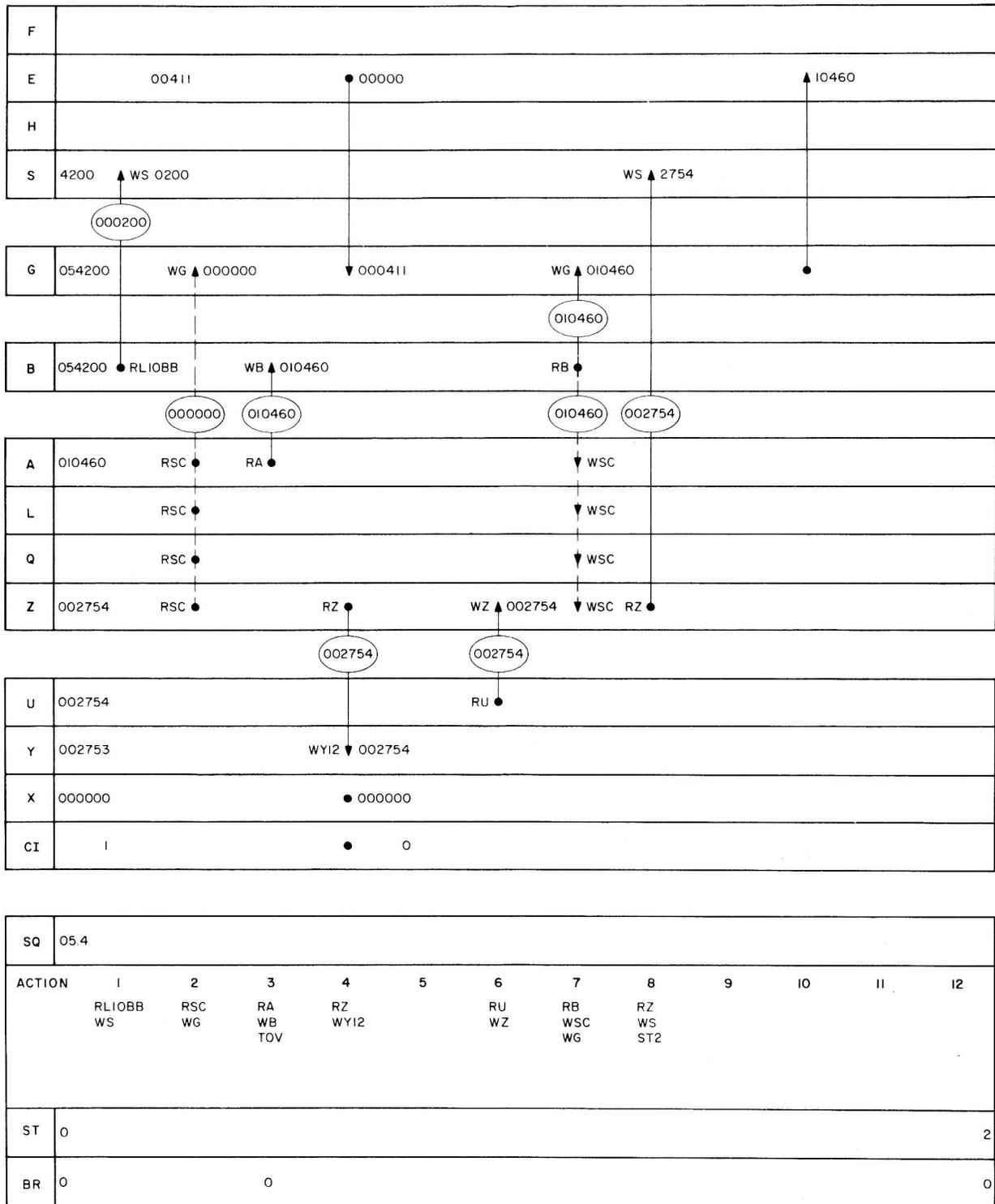
32-86. Special Cases of TS E:

- a. TS A (alternate code DVSK, for overflow skip) retains $c(A)$, including any overflow bit.
- b. TS L and TS Q are useful. Registers L and Q also store an overflow bit contained in A; however, bits 16 through 13 of $c(Z)$ are cleared during the execution of STD2.
- c. TS Z (alternate code TCAA, for transfer control to address in A) is similar to TS L and TS Q.
- d. TS EBANK, TS FBANK, and TS BBANK can be used; however, the particular write operations must be observed.
- e. TS ZERO has no purpose.
- f. Instructions TS E with $0010 \leq E \leq 0017$ follow the rules of paragraph 32-56.
- g. Instructions TS E with $0020 \leq E \leq 0023$ also follow the rules of paragraph 32-85 except that the data to be stored is edited as it is entered into location E.

32-87. When instruction TS E is executed, action 1 of subinstruction TS0 (row 13 of table 32-4) replaces the quantity contained in register S by the 10 bit address, thus erasing the quarter code contained in register S. Action 3 enters the content of register A into register B, and sets the branch flip-flops if the quantity entered contains an overflow bit. If no overflow bit is contained, actions 4 and 6 do not change the address contained in register Z. This address is entered into register S by action 8 and made available for subinstruction STD2. If an overflow bit is contained in register A at time 3, actions 4 and 6 increment by one the address contained in register Z and action 8 enters this incremented address into register S. In this case, subinstruction STD2 calls forward the instruction stored at the "second next" location.

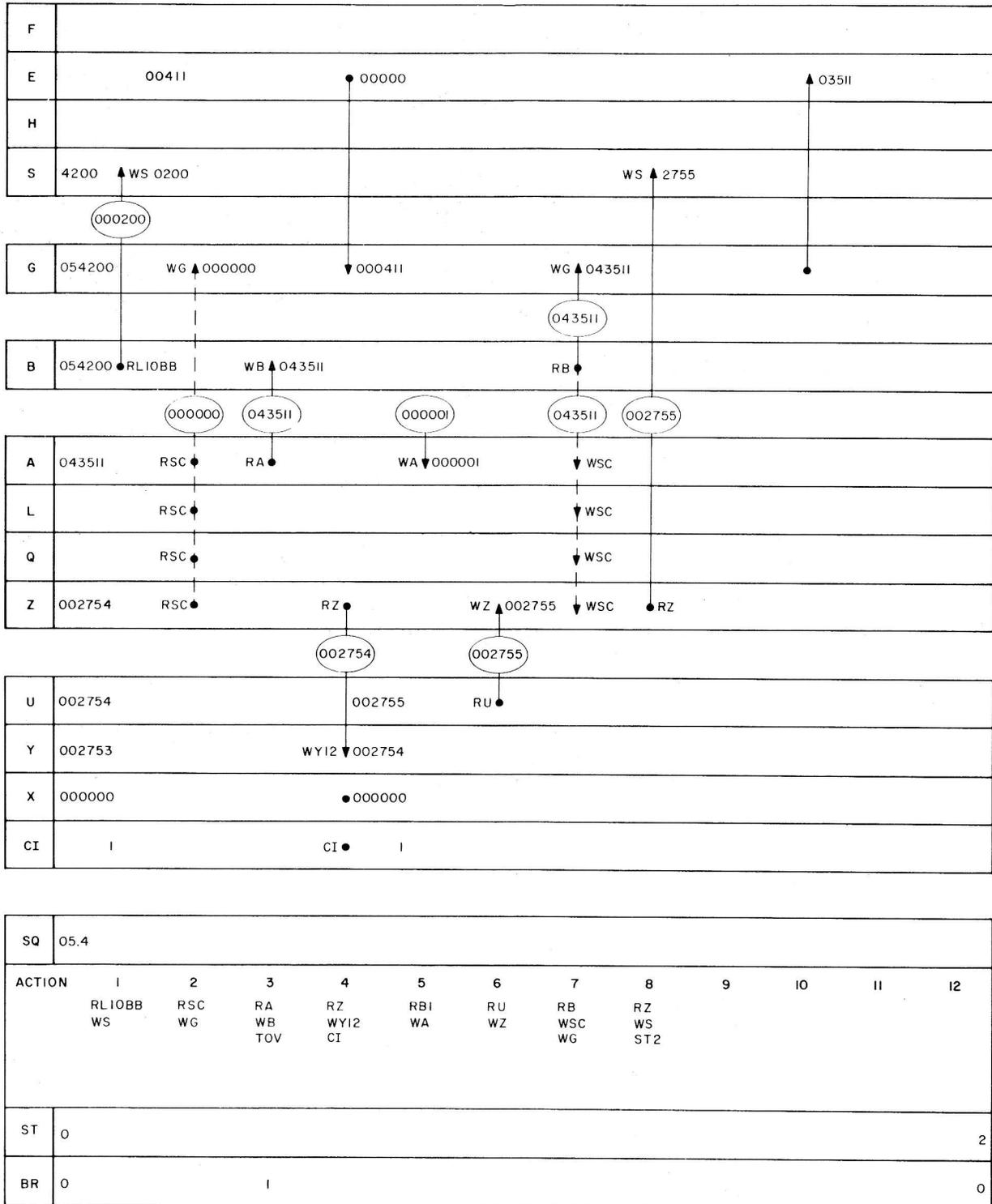
32-88. If the quantity originally contained in register A does not include an overflow bit, this quantity is copied from register B into register G by action 7. If the relevant address E of instruction TS E represents a CP register address, action 7 also copies the quantity into the addressed CP register. If address E represents an E Memory location, the quantity is entered into the addressed location at time 10.

32-89. If the quantity originally contained in register A includes an overflow bit, this quantity is also entered into register G by action 7, and into a CP register if required. Otherwise, the same quantity (but without the overflow bit) is entered into E Memory at time 10. Furthermore, action 5 replaces the overflow quantity in register A by quantity 000001 in case of positive overflow, or by 177776 in case of negative overflow.



2724A

Figure 32-12. Subinstruction TS0, Without Overflow Bit in Register A



2725A

Figure 32-13. Subinstruction TS0, With Positive Overflow Bit in Register A

32-90. The execution of subinstruction TS0 of TS 200, with A containing no overflow bit, is illustrated in figure 32-12. Figure 32-13 illustrates the execution of the same instruction when A contains a quantity with positive overflow. In case of negative overflow, control pulse R1C replaces pulse RB1 of action 5.

32-91. INSTRUCTION XCH E

32-92. Instruction XCH E (Exchange A and E) is a Basic Instruction which is represented by order code 05.6 and a 10 bit address. Instruction XCH E consists of subinstructions XCH0 and STD2, the execution of which takes two MCT's.

32-93. Instruction XCH E exchanges the data contained in register A with the data stored at location E of E Memory (or in a CP register). The operation XCH E with $0024 \leq E \leq 1777$ can be formulated as follows:

- (1) Set $c(A) = b(E)$
Set $c(E) = b(A)$ except the overflow bit which is lost.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction XCH E, and j being the instruction stored at location (I+1).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(I+1) = b(I+1)$ if (I+1) represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-94. Special Cases of XCH E:

- a. XCH A has no purpose.
- b. XCH L, XCH Q, and XCH Z exchange data without losing an overflow bit.
- c. XCH EBANK, XCH FBANK, and XCH BBANK can be used, but the particular read and write operations must be observed.
- d. XCH ZERO sets $c(A) = 000000$.
- e. Instructions XCH E with $0010 \leq E \leq 0017$ follow the rules of paragraph 32-93.
- f. Instructions XCH E with $0020 \leq E \leq 0023$ also follow the rules of paragraph 32-94 except that $b(A)$ is edited as it is transferred to location E.

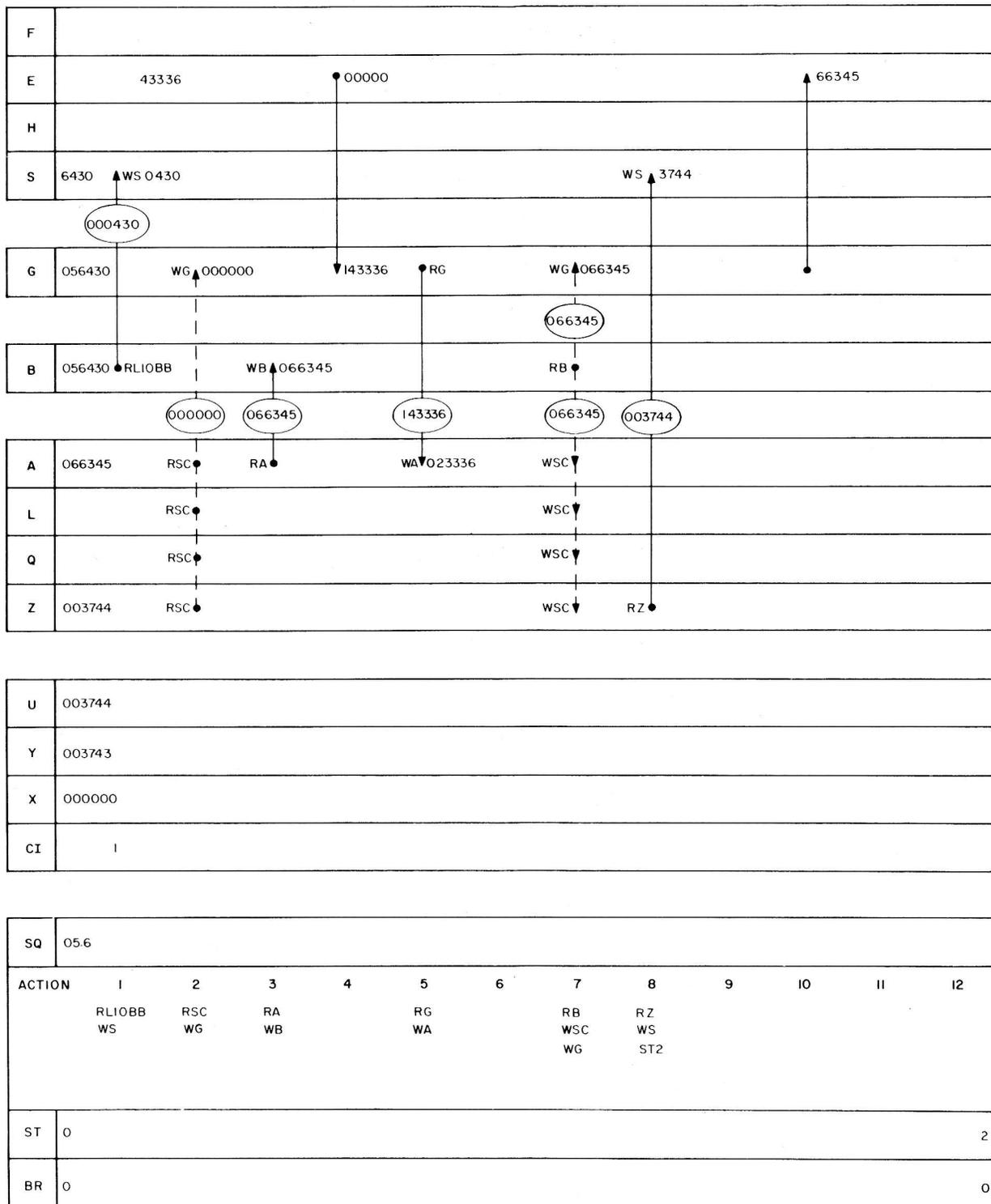


Figure 32-14. Subinstruction XCH0

32-95. When instruction XCH E is executed, action 1 of subinstruction XCH0 (row 14 of table 32-3) replaces the quantity contained in register S by the 10 bit address, thus erasing the quarter code contained in register S. The quantity stored at location E is entered into register G at time 2 or 4 and into register A by action 5. The quantity originally contained in register A is entered into register B by action 3 and transferred to location E at time 7. Action 8 enters the address of the next instruction into register S and subinstruction STD2 calls forward the next instruction as usual.

32-96. The execution of subinstruction XCH0 of XCH 0430 is illustrated in figure 32-14. The overflow bit originally contained in register A is lost on the way to location 0430 in E Memory. The sign bit originally contained in location 0430 moves into bit position 16 as the quantity is entered into register G.

32-97. INSTRUCTION LXCH E

32-98. Instruction LXCH E (Exchange L and E) is a Basic Instruction which is represented by order code 02.2 and a 10 bit address. Instruction LXCH E consists of subinstructions LXCH0 and STD2, the execution of which takes two MCT's.

32-99. Instruction LXCH E exchanges the data contained in register L with the data stored at location E of E Memory (or in a CP register). The operation LXCH E with $0024 \leq E \leq 1777$ can be formulated as follows:

- (1) Set $c(L) = c(E)$.
Set $c(E) = b(L)$ except the overflow bit which is lost.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction LXCH E, and j being the instruction stored at location (I+1).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(I+1) = b(I+1)$ if (I+1) represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-100. Special Cases of LXCH E:

- a. LXCH A exchanges $c(L)$ and $c(A)$ and retains the overflow bit.
- b. LXCH L has no purpose.

- c. LXCH Q and LXCH Z also exchange data and retain the overflow bit.
- d. LXCH EBANK, LXCH FBANK, and LXCH BBANK can be used, but the particular read and write operations must be observed.
- e. LXCH ZERO (alternate code ZL, for Zero L) sets $c(L) = 000000$.
- f. Instructions LXCH E with $0010 \leq E \leq 0017$ follow the rules of paragraph 32-99.
- g. Instructions LXCH E with $0020 \leq E \leq 0023$ also follow the rules of paragraph 32-99 except that $b(L)$ is edited as it is transferred to location E.

32-101. The execution of instruction LXCH E is similar to that of instruction XCH E except for actions 3 and 5. (Compare rows 14 and 15 of table 32-4.) Subinstruction LXCH0 enters the content of register L (instead of A) into register B and enters the content of register G into register L instead of into A.

32-102. INSTRUCTION QXCH E

32-103. Instruction QXCH E (Exchange Q and E) is an Extra Code Instruction which is represented by order code 12.2 and a 10 bit address. Instruction QXCH E must be preceded by Special Instruction EXTEND which enters a ONE into bit position EXT of register SQ. (Code 2.2 is taken from bit positions 16 and 14 through 11 of register B and entered into the corresponding bit positions of register SQ.) Instruction QXCH E consists of subinstructions QXCH0 and STD2, the execution of which takes two MCT's.

32-104. Instruction QXCH E exchanges the data contained in register Q with the data stored at location E of E Memory (or in a CP register). The operation QXCH E with $0024 \leq E \leq 1777$ can be formulated as follows:

- (1) Set $c(Q) = c(E)$
Set $c(E) = b(Q)$ except the overflow bit which is lost.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction LXCH E, and j being the instruction stored at location (I+1).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(I+1) = b(I+1)$ if (I+1) represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-105. Special Cases of QXCH E:

- a. QXCH A exchanges $c(Q)$ and $c(A)$ and retains the overflow bit.
- b. QXCH L exchanges $c(Q)$ and $c(L)$ and retains the overflow bit.
- c. QXCH Q has no purpose.
- d. QXCH Z exchanges $c(Q)$ and $c(Z)$.
- e. QXCH EBANK, QXCH FBANK, and QXCH BBANK can be used, but the particular read and write operations must be observed.
- f. QXCH ZERO (alternate code ZQ, for Zero Q) sets $c(Q) = 000000$.
- g. Instructions QXCH E with $0010 \leq E \leq 0017$ follow the rules of paragraph 32-104.
- h. Instructions QXCH E with $0020 \leq E \leq 0023$ also follow the rules of paragraph 32-104 except that $b(Q)$ is edited as it is transferred to location E.

32-106. The execution of instruction QXCH E is similar to that of instructions XCH E and LXCH E except for actions 3 and 5. (Compare rows 14, 15, and 16 of table 32-4.) Subinstruction QXCH0 takes data from and enters data into register Q instead of register A.

32-107. INSTRUCTION DXCH E

32-108. Instruction DXCH E (Double Exchange A and E) is a Basic Instruction which is represented by order code 05.2 and a 10 bit address. DXCH E consists of subinstructions DXCH0, DXCH1, and STD2, the execution of which takes three MCT's.

32-109. Instruction DXCH E exchanges the double precision quantity contained in registers A and L with the double precision quantity stored at locations E and (E+1) of E Memory (or in two CP registers). The operation DXCH E with $0024 \leq E \leq 1776$ excluding the last address of any E memory bank (table 30-2) can be formulated as follows:

- (1) Set $c(A) = c(E)$.
 Set $c(L) = c(E+1)$.
 Set $c(E) = b(A)$ }
 Set $c(E+1) = b(L)$ } except any overflow bit which is lost.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction DXCH E,
 and j being the instruction stored at location (I+1).
 Set $c(S) =$ relevant address of j.
 Set $c(SQ) =$ order code of j

- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(I+1) = b(I+1)$ if $(I+1)$ represents an address in E Memory.

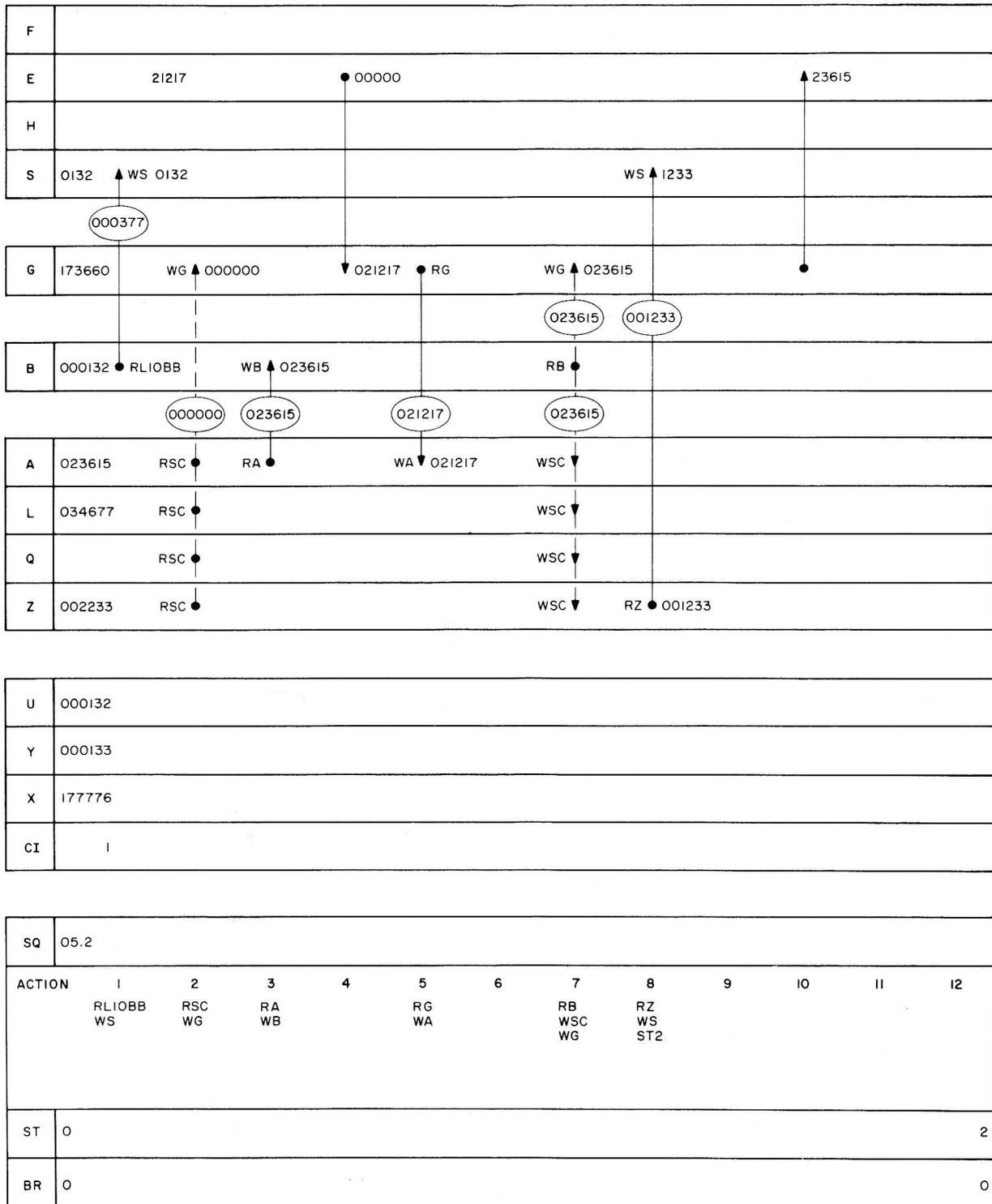
Point (2) implies that instruction j is executed next.

32-110. Special cases of DXCH E:

- a. DXCH A has no purpose
- b. DXCH 0010 (DXCH ARUPT) and DXCH 0013 (table 30-4) are useful and follow the rules of paragraph 32-110.
- c. DXCH FBANK (alternate code DTCTF for double precision transfer control fixed bank) and DXCH Z (alternate code DTCTB for double precision transfer control both banks) can be used to change the content of register Z and of a bank register. These double precision transfer control instructions can be used to "jump" banks and store a return address plus its bank code in registers A and L.
- d. Any DXCH E with $0000 \leq E \leq 0022$ must be used with extreme care so as not to destroy stored data; E must not be 0023 in order to prevent destruction of data in counter T2. Whenever locations 0020 through 0023 are involved, a quantity entered into any of these locations is edited.

32-111. The execution of instruction DXCH E is similar to the execution of instructions DCA K, XCH E, and LXCH E. When instruction DXCH E is executed, first the contents of register L and location $(E+1)$ are exchanged by subinstruction DXCH0, then the contents of register A and location E are exchanged by subinstruction DXCH1. The Yul Programming System accomplishes this by replacing instruction DXCH E with code DXCH $(E+1)$ which is wired into the program. As the AGC executes subinstruction DXCH0, relevant address $(E+1)$ (available first) is decremented by one. Subinstruction DXCH1 then uses the decremented address E. For execution of subinstructions DXCH0 and DXCH1 refer to rows 17 and 18 of table 32-4. When double precision information is exchanged with memory, E must not be equal to the last address of any E memory bank to allow $E+1$ to be the next address in the same bank.

32-112. The execution of instruction DXCH 0132 is illustrated in figures 32-15 and 32-16. Location 0132 contains quantity 21217 and location 0133 contains quantity 34677.



2723A

Figure 32-16. Subinstruction DXCH1

32-113. MODIFYING INSTRUCTIONS

32-114. INSTRUCTION NDX E

32-115. Instruction NDX E (Index Next Basic Instruction with E) is a Basic Instruction which is represented by order code 05.0 and a 10 bit address. The alternate spelling for NDX E is INDEX E. Instruction NDX E consists of subinstructions NDX0 and NDX1, the execution of which takes two MCT's.

32-116. Instruction NDX E takes as the next instruction the arithmetic sum of the instruction located at the next location plus the quantity stored at location E. The operation NDX E with $0024 \leq E \leq 1777$ can be formulated as follows:

- (1) Derive a new Basic Instruction (j) by adding the $c(E)$ to the $c(I+1)$. The address of location $I+1$ is initially contained in Z, I being the address of instruction NDX E.
- (2) Set $c(B) = j$.
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(E) = b(E)$ and $c(I+1) = b(I+1)$ if E and/or $(I+1)$ represent an address in E Memory.

Point (2) implies that instruction j is executed next.

32-117. Special Cases of NDX E:

- a. NDX A, NDX L, NDX Q, and NDX Z are useful. Registers A, L, Q, and Z are able to store a 4 bit order code besides a 12 bit address.
- b. NDX EBANK, NDX FBANK, and NDX BBANK may be used; however, the particular read and write operations must be observed.
- c. NDX ZERO has no purpose.
- d. Instructions NDX E with $0010 \leq E \leq 0016$ follow the rules of paragraph 32-116.
- e. NDX 0017 = RESUME. See paragraph 32-271.
- f. Instructions NDX E with $0020 \leq E \leq 0023$ also follow the rules of paragraph 32-116 except that $c(E)$ is edited during restoring.

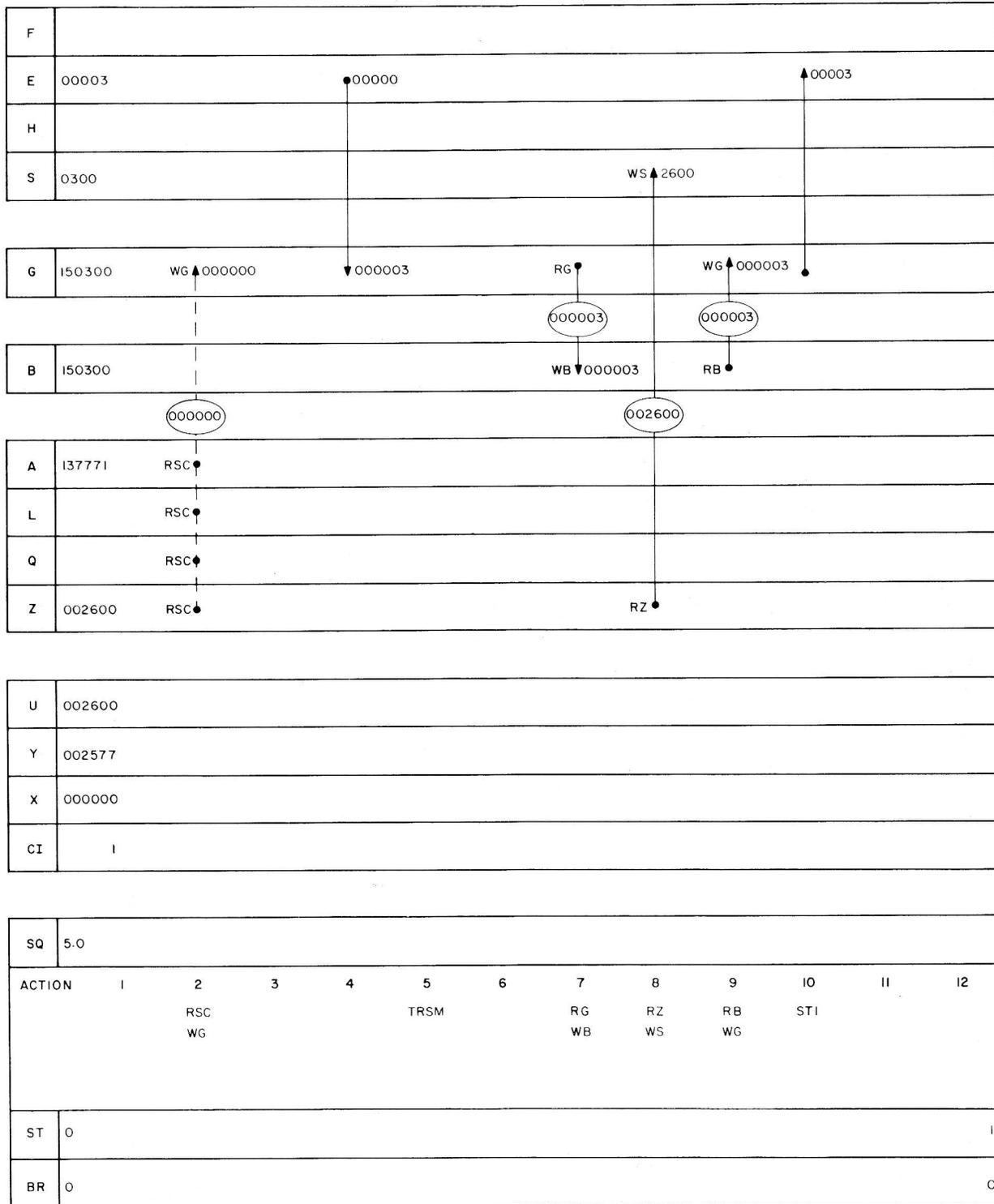
32-118. The instruction derived by instruction NDX E may be similar to the instruction stored at location (I+1) or may be quite different. If the quantity at location E is equal to or smaller than the complement of the relevant address stored in location (I+1), then the order code of the new instruction is equal to the order code of the instruction contained at (I+1). If the quantity at location E is larger, then the order code of the new instruction differs from the order code at (I+1). For example, if NDX E with $c(E) = 100$ is followed by TC 2000, then the new instruction becomes TC 2100. If $c(E) = 5777$, then the new instruction is TC 7777. However, if $c(E) = 6000$ or larger, the new instruction is CCS 0000, or any instruction other than TC K. In general, the new instruction can be expressed as $c(I+1), c(E)$. If the quantity at location E is equal to or small than the complement of the relevant address K of instruction OC K stored at location I+1, then the new instruction can be expressed as $OC[K+c(E)]$ where OC stands for order code. The derived instruction is always a Basic Instruction, not an Extra Code Instruction as explained in paragraphs 32-125 and 32-126.

32-119. When instruction NDX E is executed, subinstruction NDX0 (row 19 of table 32-4) enters the quantity stored at location E into registers G and B and enters the address of the location following instruction NDX E (address I+1 stored in register Z) into register S. Subinstruction NDX1 (row 20) then enters the instruction from location I+1 into the Adder, together with the quantity from location E, and uses the sum as the next instruction. (The quantity from location E is moved from register B via registers Z and A into register X by actions 3, 5, and 7 while the quantity originally contained in register A is temporarily stored in register B by action 4 and returned to A by action 9.) Action 8 enters the relevant address of the new instruction into register S, action 10 enters the new instruction into register B, and action 12 enters its order code into register SQ.

32-120. The execution of instruction NDX 0300 is illustrated in figures 32-17 and 32-18. It is assumed that the instruction is stored at location 2577, that instruction AD 0420 is stored at location 2600, and that the indexing quantity 00003 is stored at location 0300. The modified instruction is as follows: $AD[0420+c(0300)] = AD 0423$.

32-121. INSTRUCTION NDX K

32-122. Instruction NDX K (Index Next Extra Code Instruction with K) is an Extra Code Instruction which is represented by order code 15 and a 12-bit address. The alternate spelling for NDX K is INDEX K. Instruction NDX K must be preceded by Special Instruction EXTEND which enters a ONE into bit position EXT of register SQ. Instruction NDX K consists of subinstructions NDXX0 and NDXX1, the execution of which takes two MCT's.



2720A

Figure 32-17. Subinstruction NDX0

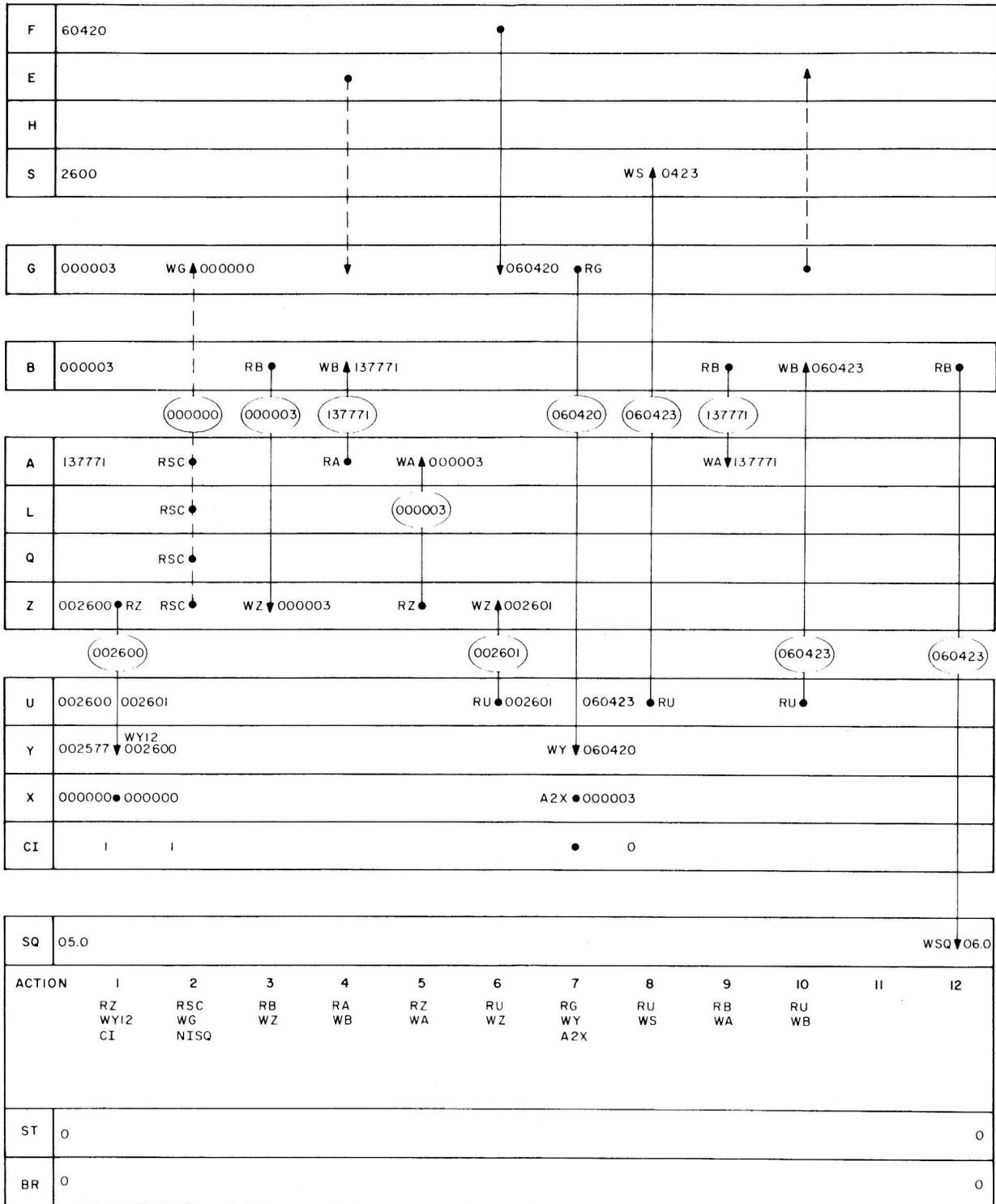


Figure 32-18. Subinstruction NDX1

32-123. Instruction NDX K takes as the next instruction the arithmetic sum of the instruction located at the next location plus the quantity stored at location E. The operation NDX K with $0024 \leq K \leq 7777$ can be formulated as follows:

- (1) Derive a new Extra Code Instruction (j) by adding $c(K)$ to $c(I+1)$. The address of location (I+1) is initially contained in Z, I being the address of instruction NDX K.
- (2) Set $c(B) = j$.
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
Set $c(EXT) = ONE$.
- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(E) = b(E)$ and $c(I+1) = b(I+1)$ if E and/or (I+1) represent an address in E Memory.

Point (2) implies that instruction j is executed next.

32-124. Special Cases of NDX K:

- a. NDX A, NDX L, NDX Q, and NDX Z are useful. Registers A, L, Q, and Z are able to store a 4-bit order code beside a 12 bit address.
- b. NDX EBANK, NDX FBANK, and NDX BBANK may be used; however, the particular read and write operations must be observed.
- c. NDX ZERO has no purpose.
- d. Instructions NDX K with $0010 \leq K \leq 0017$ follow the rules of paragraph 32-124.
- e. Instructions NDX K with $0020 \leq K \leq 0023$ also follow the rules of paragraph 32-124 except that $c(K)$ is edited during restoring.

32-125. Instructions NDX E (Index Next Basic Instruction) and NDX K (Index Next Extra Code Instruction) are similar. In table 32-4 compare rows 21 and 22 with rows 19 and 20. The main difference is that action 10 of subinstruction NDXX1 re-enters a ONE into bit position EXT of register SQ while action 10 of subinstruction NDX1 does not enter a ONE. A programmer is not concerned with which of the two instructions he should use; for this reason both instructions can be represented by the same mnemonic code, i. e., NDX or INDEX. The Yul Programming System automatically enters the proper instruction into the program.

32-126. The instruction derived by an NDX instruction is of the same type as the instruction stored after the NDX instruction, i. e. , the derived and the following instructions are both Basic Instructions or Extra Code Instructions. The EXT bit cannot be generated by the addition of action 7 of the second subinstruction. Basic Instructions can be indexed with a quantity stored in E Memory (or a CP register) only except location 0017. Extra Code Instructions can be indexed with a quantity stored anywhere in memory, including location 0017. (Action 5 of subinstruction NDX0 does test for address 0017, subinstruction NDXX0 does not.)

32-127. ARITHMETIC AND LOGIC INSTRUCTIONS

32-128. INSTRUCTION AD K

32-129. Instruction AD K (Add K) is a Basic Instruction which is represented by order code 06. and a 12 bit address. Instruction AD K consists of subinstructions AD0 and STD2, the execution of which takes two MCT's.

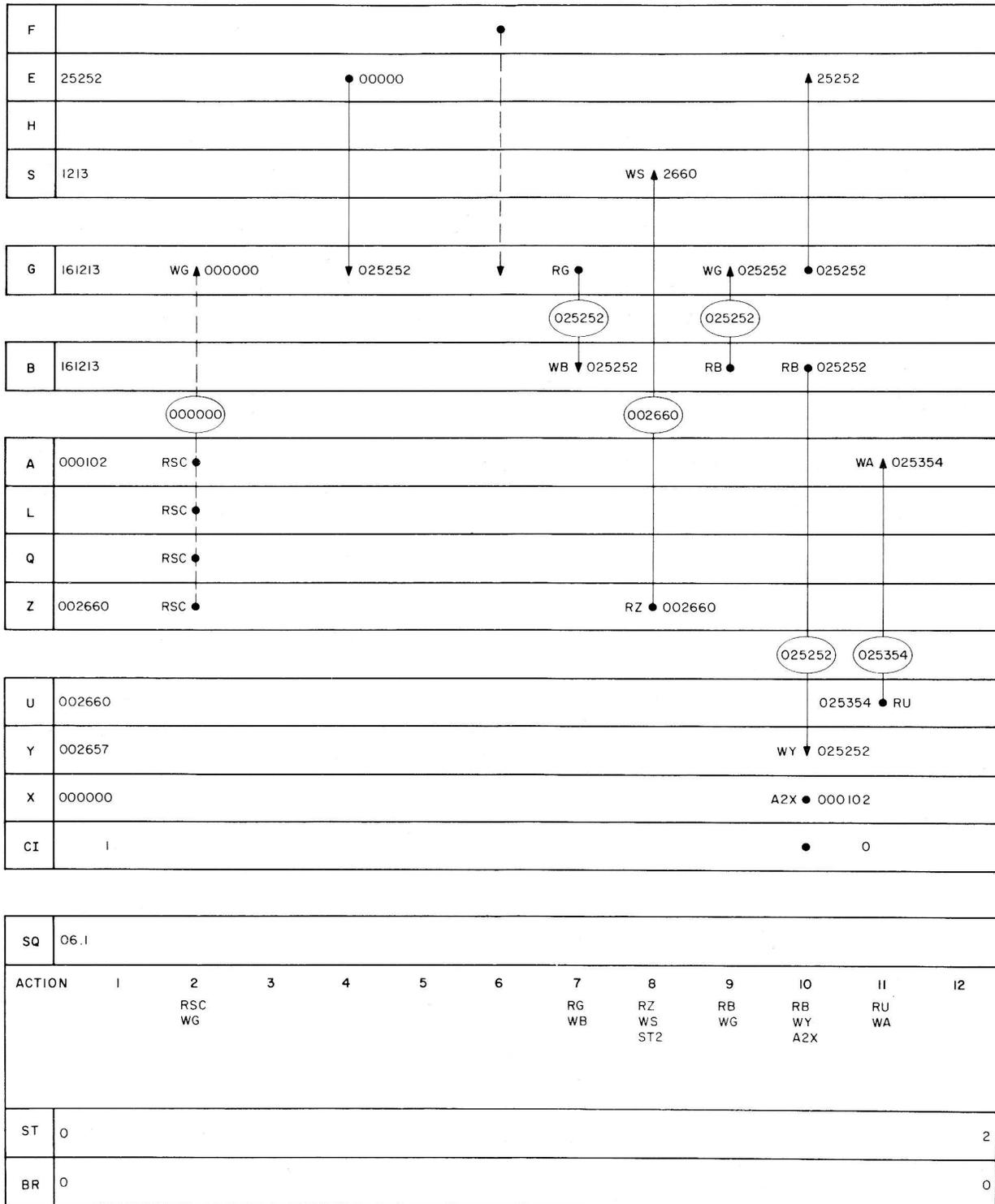
32-130. Instruction AD K adds the content of location K to the content of register A. The operation AD K with $0024 \leq K \leq 7777$ can be formulated as follows:

- (1) Set $c(A) = b(A) + c(K)$. When A and/or K contains an overflow bit, the result may be erroneous.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction AD K, and j being the instruction stored at location (I+1).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z) + 1 = I + 2$.
- (4) Restore $c(K) = b(K)$ and $c(I+1) = b(I+1)$ if K and/or (I+1) represent an address in E Memory.

Point (2) implies that instruction j is executed next.

32-131. Special Cases of AD K:

- a. AD A (alternate code DOUBLE) doubles the content of A.
- b. AD L, AD Q, and AD Z are useful.
- c. AD EBANK, AD FBANK, and AD BBANK can be used, however, the particular read and write operations must be observed.
- d. AD ZERO has no purpose.
- e. Instructions AD K with $0010 \leq K \leq 0017$ follow the rules of paragraph 32-130.



2727A

Figure 32-19. Subinstruction AD0

- f. Instructions AD K with $0020 \leq K \leq 0023$ also follow the rules of paragraph 32-130 except that the sum is edited as it is entered into K.

32-132. When instruction AD K is executed, the quantity from location K is entered into register G at time 2, 4, or 6 of subinstruction AD0 (row 23 of table 32-4). Action 7 enters the quantity into register B and action 9 re-enters the quantity into register G for restoring in E Memory at time 10. Action 10 enters the quantities in register B and register A into the Adder and action 11 transfer the sum to register A. Action 8 takes the address of the next instruction from register Z and enters it into register S. Subinstruction STD2 then calls forward the next instruction and increments by one the content of register Z as usual.

32-133. Figure 32-19 illustrates the execution of subinstruction AD0 of instruction AD 1213 stored at location 2657. This is the first subinstruction of the example discussed in paragraph 32-26. Location 1213 contains quantity 25252 and register A the quantity 000102. The sum finally provided is 025354.

32-134. INSTRUCTION SU E

32-135. Instruction SU E (Subtract E) is an Extra Code Instruction which is represented by order code 16.0 and a 10 bit address. Instruction SU E must be preceded by Special Instruction EXTEND which enters a ONE into bit position EXT of register SQ. Instruction SU E consists of subinstructions SU0 and STD2, the execution of which takes two MCT's.

32-136. Instruction SU E subtracts the content of location E from the content of register A. The operation SU E with $0024 \leq E \leq 1777$ can be formulated as follows:

- (1) Set $c(A) = b(A) + \bar{c}(E)$. When A and/or E contains an overflow bit, the result may be erroneous.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction SU E, and j being the instruction stored at location (I+1).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z) + 1 = I + 2$.
- (4) Restore $c(K) = b(K)$ and $c(I+1) = b(I+1)$ if K and/or (I+1) represent an address in E Memory.

Point (2) implies that instruction j is executed next.

32-137. Special Cases of SU E:

- a. SU A clears register A.
- b. SU L, SU Q, and SU Z are useful.
- c. SU EBANK, SU FBANK, and SU BBANK can be used, but the particular read and write functions must be observed.
- d. SU ZERO has no purpose except replacing 000000 in A by 177777.
- e. Instructions SU E with $0010 \leq E \leq 0017$ follow the rules of paragraph 32-136.
- f. Instructions SU E with $0020 \leq E \leq 0023$ also follow the rules of paragraph 32-136 except that the difference is edited as it is entered into E.

32-138. The execution of instruction SU E is very similar to that of instruction AD K. (Compare rows 23 and 24 of table 32-4.) Control pulse RB of action 10 of subinstruction AD0 is replaced by control pulse RC for subinstruction SU0.

32-139. INSTRUCTION MP K

32-140. Instruction MP K (Multiply K) is an Extra Code Instruction which is represented by order code 17. and a 12 bit address. Instruction MP K must be preceded by Special Instruction EXTEND which enters a ONE into bit position EXT of register SQ. Instruction MP K consists of subinstructions MP0, MP1, and MP3, the execution of which takes three MCT's.

32-141. Instruction MP K multiplies the content of register A by the content stored in location K and stores the double precision result in registers A and L. The operation MP K with $0024 \leq K \leq 7777$ can be formulated as follows:

- (1) Set $c(A, L) = b(A) \times c(K)$. Sign of $c(L)$ agrees with sign of $c(A)$. When $b(A)$ and/or $b(K)$ contain an overflow bit, the result is erroneous.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction MP K and j being the instruction stored at location I+1.
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z)+1 = I+2$.

- (4) Restore $c(K) = b(K)$ and $c(I+1) = b(I+1)$ if K and/or $(I+1)$ represent an address in E Memory.

Point (2) implies that instruction j is executed next.

32-142. Special Cases of MP K.

- a. MP A (alternate code SQUARE) squares the content of register A.
- b. MP L, MP Q, and MP Z are useful.
- c. MP EBANK, MP FBANK, and MP BBANK may be useful; the particular read and write operations must be observed.
- d. MP ZERO clears registers A and L.
- e. Instructions MP K with $0010 \leq K \leq 0017$ follow the rules of paragraph 32-141.
- f. Instructions MP K with $0020 \leq K \leq 0023$ also follow the rules of paragraph 32-141 (the content of K is not edited when being restored).

32-143. Principle of Operation

32-144. A multiplication as performed by instruction MP K is carried out in TWO's complement arithmetic and in a way similar to a multiplication done manually with quaternary numbers (to the base four). Let us first assume that a positive 16 bit number as used in the CP of the AGC is to be multiplied by another positive 16 bit number. The first quantity (a_8) may be 016344, the other, (k_8) 010101 when expressed in octal numbers. The two quantities can also be expressed in binary (a_2, k_2) or quaternary (a_4, k_4) form as shown in figure 32-20.

32-145. Multiplying quantity (k_4) by (a_4), starting with the lowest order digit of (a_4), the quantity zero can be taken first, or added to the starting quantity zero. At step two, the quantity (k_4) must be moved one place to the left and added to the subtotal (zero in the given example). At step three, the quantity ($2k_4$) has to be added in a similar way because the third last digit of (a_4) is 2. In the fourth step, the quantity ($3k_4$) could be added; however, subtracting the quantity (k_4) and adding the quantity (k_4) at step five has the same effect ($1 \times 4^{N+1} - 1 \times 4^N = 3 \times 4^N$). At step six, the quantity (k_4) is subtracted for the same reason as in step four because the sixth lowest digit is a 3 like the fourth lowest. At step seven, the quantity ($2k_4$) instead of (k_4) must be added to make up for the carry over of step six. The product has been translated into binary and octal notation. To prove the result, the multiplication (a)x(k) has been carried out with octal numbers in the simplest way. Both results agree as shown.

32-146. In the second example, figure 32-21, (a) is a positive quantity and (k) is a negative quantity. The quantity (k) is the TWO's complement of quantity (k) of the first example. The correctness of the second example can be proved by comparing its results with those of the first examples. The binary, quaternary, and octal products of the second example are all TWO's complement numbers of the first example.

32-147. Adding 0's in front of a positive TWO's complement quaternary number does not change its value; adding 3's (minus zeroes) in front of a negative number does not change its value. Adding 0's at the end of a positive or a negative TWO's complement quaternary number does not change its value.

32-148. Actual Execution

32-149. When instruction MP K is executed, the quantity from location K is entered into register G at time 2, 4, or 6 of subinstruction MP0 (row 25 of table 32-4 and figures 32-22 through 32-25). The quantity from K is later used as the multiplicand (k). Action 3 enters the quantity from register A into register B; this quantity is used as the multiplier (a). Action 4 enters the multiplier (a) always in its positive form into register L. Action 7 enters the multiplicand (k) into register B. The branch flip-flops have been set by actions 3 and 7 as stated by notes $\triangle 3$ through $\triangle 6$ in the last column of row 25 in table 32-4. According to this setting of the branch flip-flops, the multiplicand (k) is re-entered by actions 9 and 10 into register B either in its positive or in its negative TWO's complement form.

32-150. After time 10, register L always contains the multiplier (a) in its positive form. Register B contains the multiplicand (k) in its positive form if (a) and (k) have the same sign, or in its negative form if (a) and (k) have opposite signs. In case of equal signs, register A is set to zero at time 11 and later accumulates the product. In case of different signs, the quantity 177777 (minus one, TWO's complement) is entered into register A as a starting quantity to make the final product a ONE's complement number; furthermore, a ONE is entered into bit position 16 of register L by action 11 to indicate that the final product must be negative.

32-151. Eleven actions of subinstruction MP1 and the first three actions of subinstruction MP3 (rows 26 and 27 of table 32-4) perform the actual multiplication. In figures 32-26 and 32-27, the multiplication with multiplicand and multiplier of equal sign (figure 32-22 and figure 32-25) is continued. In figures 32-28 and 32-29 the multiplication with two quantities of opposite sign (figure 32-23 and 32-24) is continued. The same multiplications are also illustrated in figures 32-30 and 32-31 to explain the individual operations. The results of figure 32-31 are the ONE's complement of the results of figure 32-30.

$$\begin{aligned}
 a_8 &= 0 \ 1 \ 6 \ 3 \ 4 \ 4 \\
 a_2 &= 0001110011100100 \\
 a_4 &= 0 \ 1 \ 3 \ 0 \ 3 \ 2 \ 1 \ 0
 \end{aligned}$$

$$\begin{aligned}
 k_8 &= 0 \ 1 \ 0 \ 1 \ 0 \ 1 \\
 k_2 &= 0001000001000001 \\
 k_4 &= 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1
 \end{aligned}$$

$k_4 \times a_4$:

	0 0 0 0 0 0 0 0 0
$+0 \times k_4$	0 0 0 0 0 0 0 0 0

	0 0 0 0 0 0 0 0 0
$+1 \times k_4$	0 1 0 0 1 0 0 1

	0 0 1 0 0 1 0 0 1 0
$+2 \times k_4$	0 2 0 0 2 0 0 2

	0 0 2 1 0 2 1 0 2 1 0
$-1 \times k_4$	3 2 3 3 2 3 3 3

	3 3 3 2 0 3 2 0 3 2 1 0
$+1 \times k_4$	0 1 0 0 1 0 0 1

	0 0 0 3 2 1 3 2 1 3 2 1 0
$-1 \times k_4$	3 2 3 3 2 3 3 3

	3 3 3 0 3 1 1 3 1 1 3 2 1 0
$+2 \times k_4$	0 2 0 0 2 0 0 2

$k_4 \times a_4$	0 0 1 3 1 1 1 2 1 1 1 3 2 1 0
$(k \times a)_2$	0 0 0 0 0 1 1 1 0 1 0 1 0 1 1 0 0 1 0 1 0 1 1 1 1 0 0 1 0 0
$(k \times a)_8$	0 1 6 5 3 1 2 7 4 4

Proof: $a_8 \times k_8$

1	6	3	4	4					
	1	6	3	4	4				
			1	6	3	4	4		
1	6	5	3	1	2	7	4	4	

Figure 32-20. Positive Product, Principle of Multiplication

$$\begin{aligned}
 a_8 &= 0 \ 1 \ 6 \ 3 \ 4 \ 4 \\
 a_2 &= 0001110011100100 \\
 a_4 &= 0 \ 1 \ 3 \ 0 \ 3 \ 2 \ 1 \ 0
 \end{aligned}$$

$$\begin{aligned}
 k_8 &= 1 \ 6 \ 7 \ 6 \ 7 \ 7 \\
 k_2 &= 1110111110111111 \\
 k_4 &= 3 \ 2 \ 3 \ 3 \ 2 \ 3 \ 3 \ 3
 \end{aligned}$$

$k_4 \times a_4:$

	0 0 0 0 0 0 0 0
+0k ₄	0 0 0 0 0 0 0 0
	0 0 0 0 0 0 0 0
+1k ₄	3 2 3 3 2 3 3 3
	3 3 2 3 3 2 3 3 0
+2k ₄	3 1 3 3 1 3 3 2
	3 3 1 2 3 1 2 3 1 3 0
-1k ₄	0 1 0 0 1 0 0 1
	0 0 0 1 3 0 1 3 0 1 3 0
+1k ₄	3 2 3 3 2 3 3 3
	3 3 3 0 1 2 0 1 2 0 1 3 0
-1k ₄	0 1 0 0 1 0 0 1
	0 0 0 3 0 2 2 0 2 2 0 1 3 0
+2k ₄	3 1 3 3 1 3 3 2

$$\begin{aligned}
 k_4 \times a_4 & 3 \ 3 \ 2 \ 0 \ 2 \ 2 \ 2 \ 1 \ 2 \ 2 \ 2 \ 0 \ 1 \ 3 \ 0 \\
 (k \times a)_2 & 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \\
 (k \times a)_8 & 7 \ 6 \ 1 \ 2 \ 4 \ 6 \ 5 \ 0 \ 3 \ 4
 \end{aligned}$$

Figure 32-21. Negative Product, Principle of Multiplication

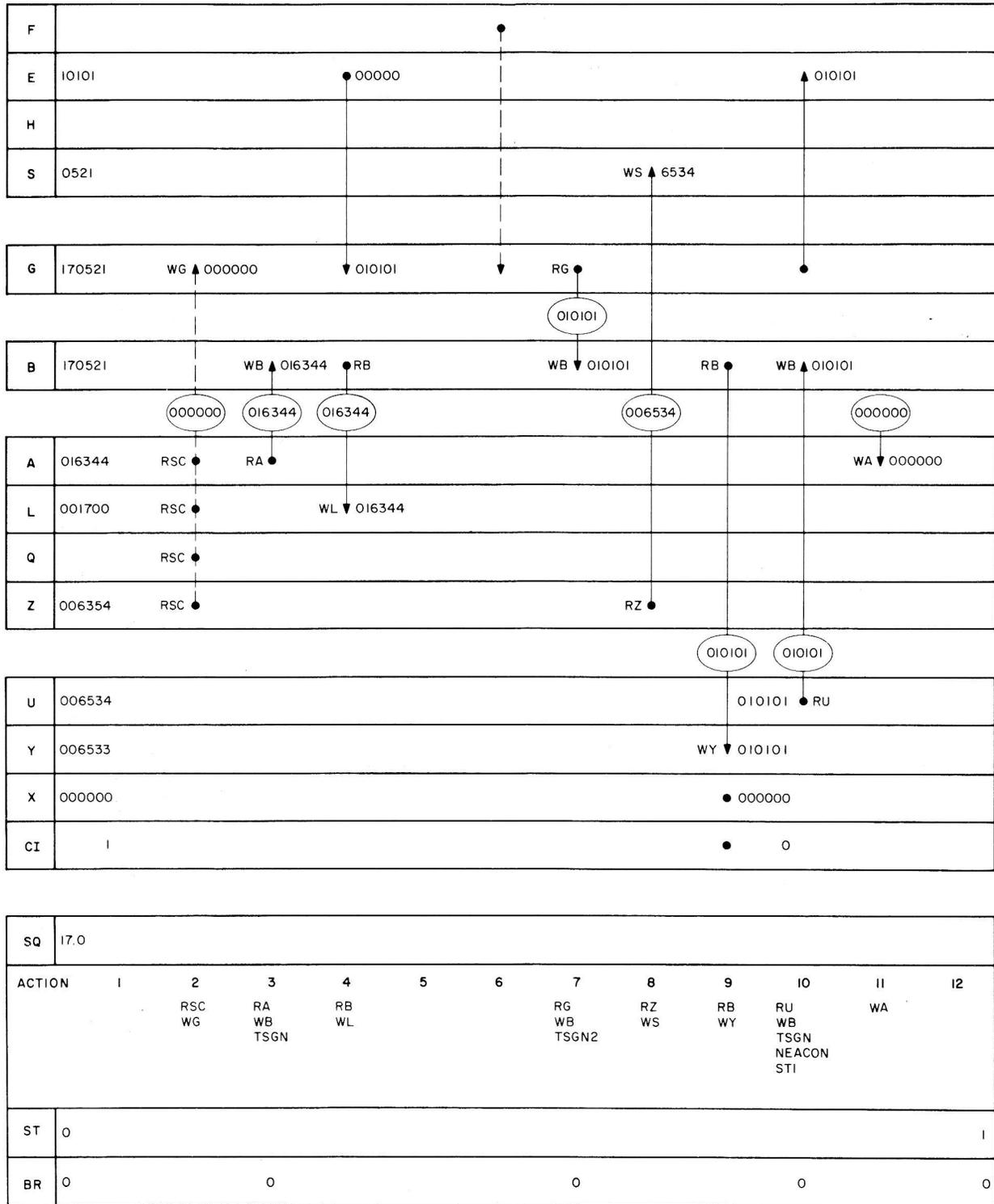
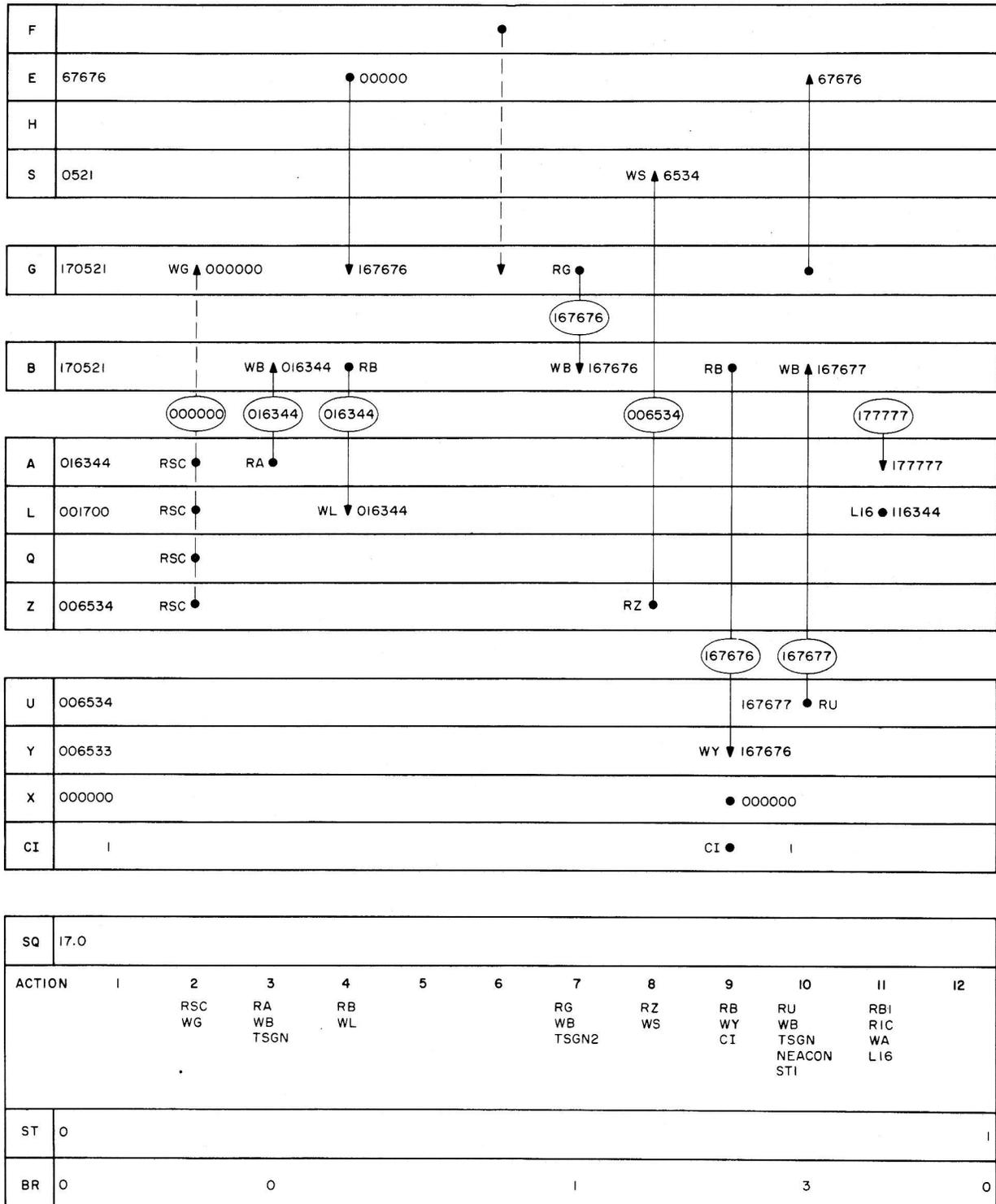


Figure 32-22. Subinstruction MP0, With Two Positive Quantities



2799A

Figure 32-23. Subinstruction MP0, With Positive Quantity in A and Negative Quantity in E

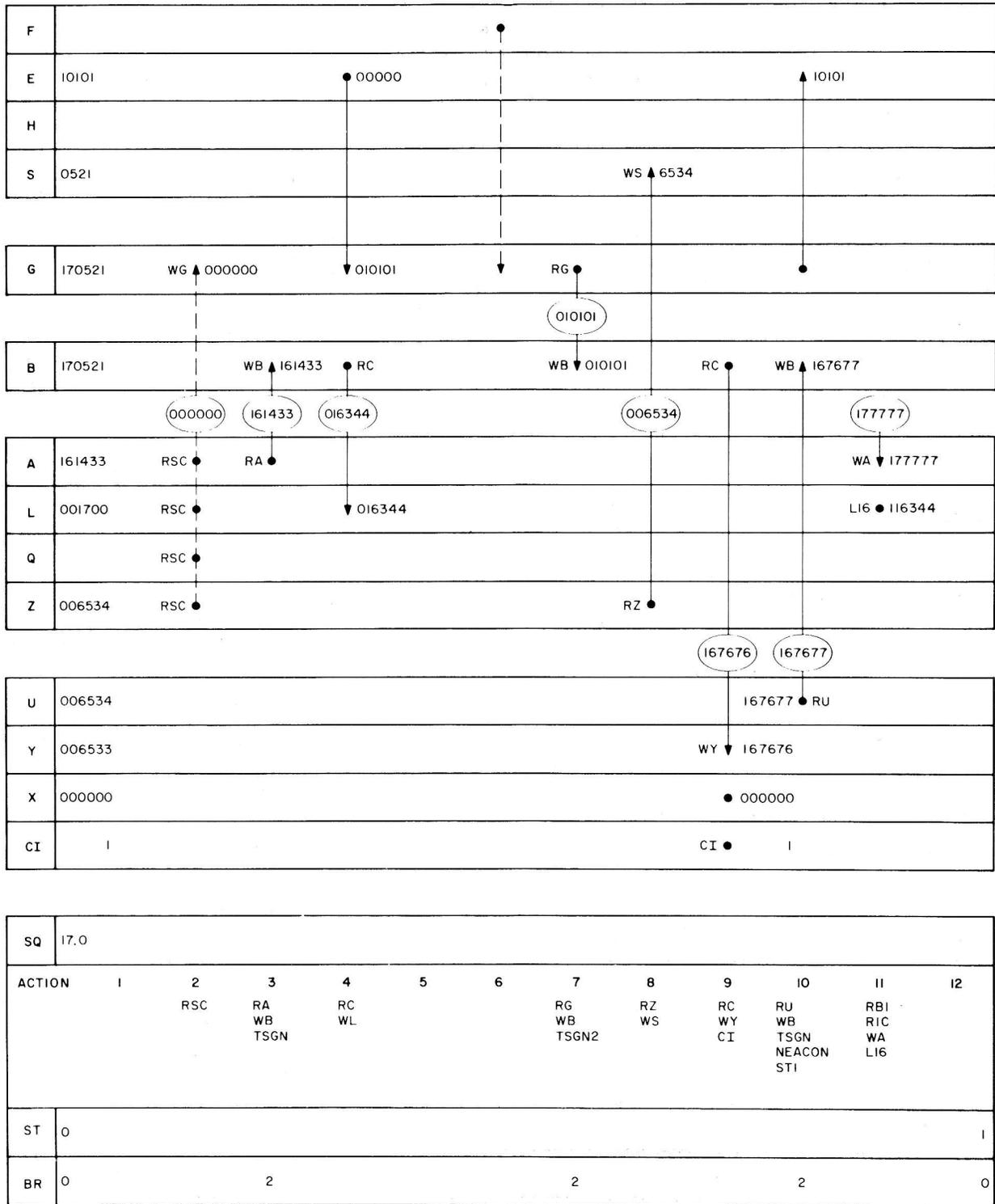
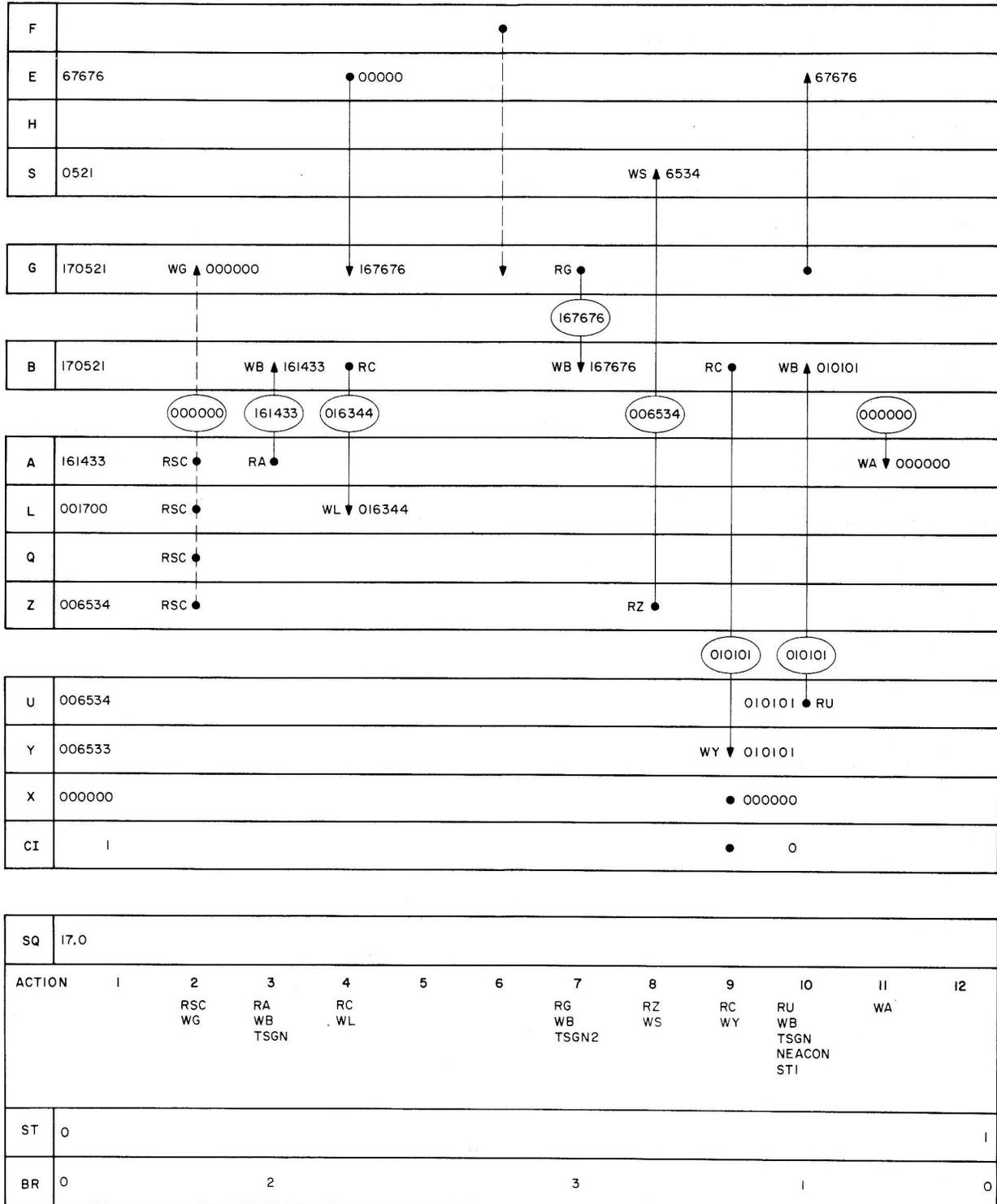


Figure 32-24. Subinstruction MP0, with Negative Quantity in A and Positive Quantity in E



2798A

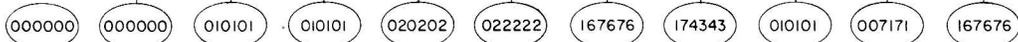
Figure 32-25. Subinstruction MPO, With Two Negative Quantities

F	
E	
H	
S	6534

G	010101	L2GD ●034710	L2GD●007162	L2GD●021634	L2GD●044347 MCRO	L2GD●071070	L2GD●036217 MCRO
---	--------	-----------------	-------------	-------------	---------------------	-------------	---------------------

B	010101	RB●	RB●	RC●	RB●	RC●
---	--------	-----	-----	-----	-----	-----

A	000000	WALS▲000000	WALS▲002020	WALS▲004444	WALS▲177070	WALS▲001636
L	016344	WALS G2LS▲003471	WALS G2LS▲01076	WALS G2LS▲022163	WALS G2LS▲074434 MCRO	WALS G2LS▲017107
Q						
Z	006534					

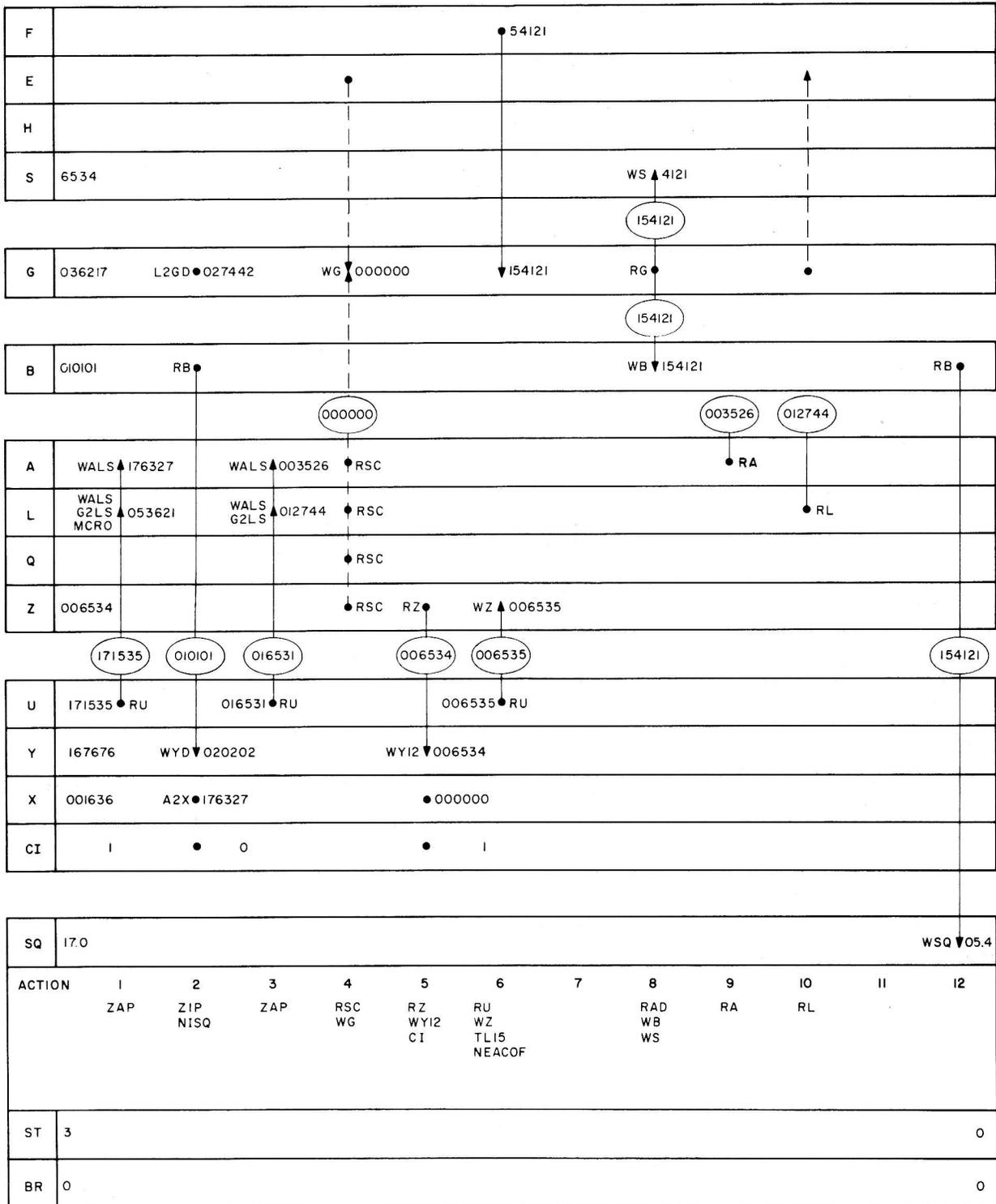


U	010101	000000 ● RU	010101 ● RU	022222 ● RU	174343 ● RU	007171 ● RU	171535
Y	010101	WY▼000000	WY▼010101	WYD▼020202	WY▼167676	WY▼010101	WY▼167676
X	000000	A2X ●000000	A2X●000000	A2X●002020	A2X●004444	A2X●177070	A2X●001636
CI	0●	0	●	0	●	0	CI● I

SQ	17.0											
ACTION	I	2	3	4	5	6	7	8	9	10	11	12
	ZIP	ZAP	ZIP	ZAP	ZIP	ZAP	ZIP	ZAP	ZIP	ZAP ST1 ST2	ZIP	
ST	I											3
BR	0											0

2801A

Figure 32-26. Subinstruction MPI, Positive Product



2802A

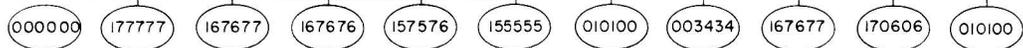
Figure 32-27. Subinstruction MP3, Positive Product

F	
E	
CH	
S	6534

G	010101 ● L2GD ● I34710	L2GD ● I67162	L2GD ● I55634	MCRO L2GD ● I33347	L2GD ● I06670	L2GD ● I44557
---	---------------------------	---------------	---------------	-----------------------	---------------	---------------

B	167677	RB ●	RB ●	RC ●	RB ●	RC ●
---	--------	------	------	------	------	------

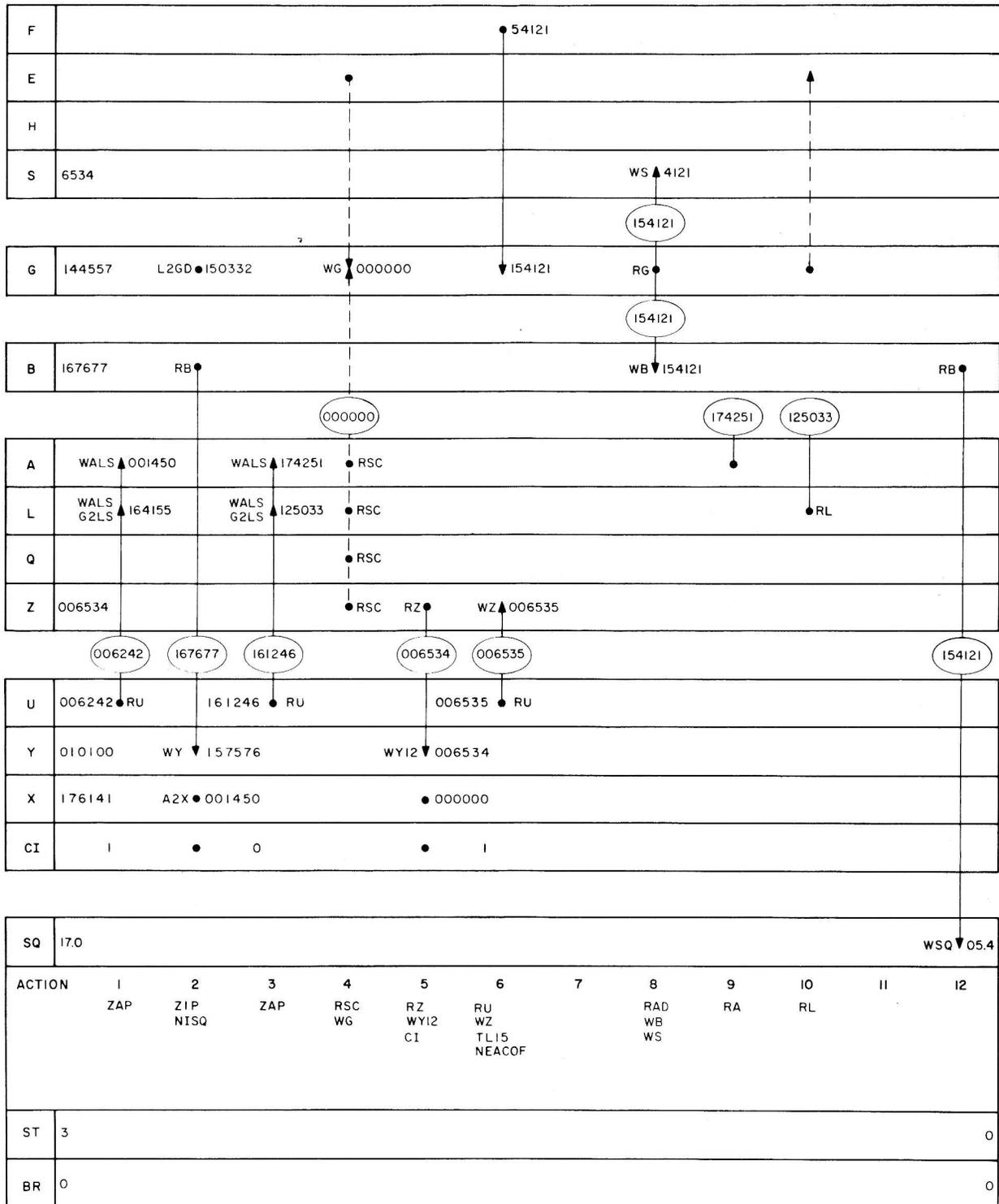
A	177777	WALS ▲ I77777	WALS ▲ I75757	WALS ▲ I73333	WALS ▲ I000707	WALS ▲ I76141
L	I16344	WALS G2LS ▲ I77777	WALS G2LS ▲ I26716	WALS G2LS ▲ I15563	WALS G2LS ▲ I03334	WALS G2LS ▲ I20667
Q						
Z	006534					



U	010101	177777 ● RU	167676 ● RU	155555 ● RU	003434 ● RU	170606 ● RU	006242
Y	010101	WY ▼ 000000	WY ▼ I67677	WYD ▼ I57576	WY ▼ 010100	WY ▼ I67677	WY ▼ 010100
X	000000 ●	A2X ● I77777	A2X ● I77777	A2X ● I75757	A2X ● I73333	A2X ● I000707	A2X ● I76141
CI	0 ●	0	● 0	● 0	CI ●	I	CI ● I

SQ	I7.0											
ACTION	1 ZIP	2 ZAP	3 ZIP	4 ZAP	5 ZIP	6 ZAP	7 ZIP	8 ZAP	9 ZIP	10 ZAP ST1 ST2	11 ZIP	12
ST	I											3
BR	0											0

Figure 32-28. Subinstruction MPI, Negative Product



2754

Figure 32-29. Subinstruction MP3, Negative Product

<u>Initial Conditions:</u>		c(A) = 016344	a ₄ = 01303210
		c(E) = 10101	
<u>After MP0:</u>		c(B) = 010101	
		c(L) = 016344	
		c(A) = 000000	
<u>Step 1</u>	L2GD	c(G) = 034710	
<u>MPI-1, 2</u>	A2X	c(X) = 000000	
	WY	c(Y) = 000000	
		c(U) = 000000	
	RU, WALS, G2LS	c(A) = 000000	c(L)=003471
<u>Step 2</u>	L2GD	c(G) = 007162	
<u>MPI-3, 4</u>	A2X	c(X) = 000000	
	RB, WY	c(Y) = 010101	
		c(U) = 010101	
	RU, WALS, G2LS	c(A) = 002020	c(L)=010716
<u>Step 3</u>	L2GD	c(G) = 021634	
<u>MPI-5, 6</u>	A2X	c(X) = 002020	
	RB, WYD	c(Y) = 020202	
		c(U) = 022222	
	RU, WALS, G2LS	c(A) = 004444	c(L)=022163
<u>Step 4</u>	L2GD, MCRO	c(G) = 044347	
<u>MPI-7, 8</u>	A2X	c(X) = 004444	
	RC, WY	c(Y) = 167676	
		c(U) = 174343	
	CI	c(CI) = 1	
	RU, WALS, G2LS, MCRO	c(A) = 177070	c(L)=074434
<u>Step 5</u>	L2GD	c(G) = 071070	
<u>MPI-9, 10</u>	A2X	c(X) = 177070	
	RB, WY	c(Y) = 010101	
		c(U) = 007171	
	RU, WALS, G2LS	c(A) = 001636	c(L)=017107

Figure 32-30. Positive Product, Actual Multiplication (Sheet 1 of 2)

Step 6	L2GD, MCRO	c(G) =	036217	
<u>MP1-11,12</u>	A2X	c(X) =	001636	
MP3-1	RC, WY	c(Y) =	167676	
		c(U) =	171535	
	CI	c(CI) =	1	
	RU, WALS, G2LS,	c(A) =	176327	c(L)=053621
	MCRO			

Step 7	L2GD	c(G) =	027442	
<u>MP3-2, 3</u>	A2X	c(X) =	176327	
	RB, WYD	c(Y) =	020202	
		c(U) =	016531	
	RU, WALS, G2LS	c(A) =	003526	c(L)=012744

Quantity in (A, L) = 165312744 as integer or
0.0725453620 as fraction

<u>Initial Conditions:</u>		c(A) = 016344	$a_4 = 01303210$
		c(E) = 67676	
<u>After MP0:</u>		c(B) = 167677	(Two's complement)
		c(L) = 116344	
		c(A) = 177777	(Two's complement minus one to make final product a ONE's complement number)
<u>Step 1</u>	L2GD	c(G) = 134710	
<u>MP1-1, 2</u>	A2X	c(X) = 177777	
	WY	c(Y) = 000000	
		c(U) = 177777	
	RU, WAL, S, G2LS	c(A) = 177777	c(L)=133471
<u>Step 2</u>	L2GD	c(G) = 167162	
<u>MP1-3, 4</u>	A2X	c(X) = 177777	
	RB, WY	c(Y) = 167677	
		c(U) = 167676	
	RU, WAL, S, G2LS	c(A) = 175757	c(L)=126716
<u>Step 3</u>	L2GD	c(G) = 155634	
<u>MP1-5, 6</u>	A2X	c(X) = 175757	
	RB, WYD	c(Y) = 157576	
		c(U) = 155555	
	RU, WAL, S, G2LS	c(A) = 173333	c(L)=115563
<u>Step 4</u>	L2GD, MCRO	c(G) = 133347	
<u>MP1-7, 8</u>	A2X	c(X) = 173333	
	RC, WY	c(Y) = 010100	
	CI	c(CI) = 1	
		c(U) = 003434	
	RU, WAL, S, G2LS, MCRO	c(A) = 000707	c(L)=103334
<u>Step 5</u>	L2GD	c(G) = 106670	
<u>MP1-9, 10</u>	A2X	c(X) = 000707	
	RB, WY	c(Y) = 167677	
		c(U) = 170606	
	RU, WAL, S, G2LS	c(A) = 176141	c(L)=120667

Figure 32-31. Negative Product, Actual Multiplication (Sheet 1 of 2)

<u>Step 6</u>	L2GD, MCRO	c(G) =	144557	
MP1-11	A2X	c(X) =	176141	
MP3-1	RC, WY	c(Y) =	010100	
	CI	c(CI) =	1	
		c(U) =	006242	
	RU, WAL, G2LS, MCRO	c(A) =	001450	c(L)=164155
<u>Step 7</u>	L2GD	c(G) =	150332	
MP3-2, 3	A2X	c(X) =	001450	
	RB, WYD	c(Y) =	157576	
		c(U) =	161246	
	RU, WAL, G2LS	c(A) =	174251	c(L)=125033

Quantity in (A, L)=612465033 as integer or
.7052324157 as fraction

32-152. Actions 4 through 12 of subinstruction MP3 conclude the operation of instruction MP K. At time 4 of 6, the next instruction is called forward from E or F Memory, respectively, action 8 enters the relevant address into register S and the whole instruction into register B. At time 12, the order code is entered into register SQ. Actions 5 and 6 increment by one the content of register Z. All these operations are normally performed by subinstruction STD2.

32-153. In case bit position 15 of register L contains a ONE at time 6 of subinstruction MP3, indicating that a carry over from the last step remained, the multiplicand is once more added to the product by actions 7 and 11.

32-154. INSTRUCTION DV E

32-155. Instruction DV E (Divide by E) is an Extra Code Instruction which is represented by order code 11.0 and a 10 bit address. Instruction DV E must be preceded by Special Instruction EXTEND which enters a ONE into bit position EXT of register SQ. Instruction DV E consists of subinstructions DV0, DV1, DV3, DV7, DV6, DV4, and STD2, the execution of which takes six MCT's. Subinstruction DV0 has only three actions (1 through 3); subinstruction DV1, DV3, DV7, and DV6 each have 12 actions (4 through 12, 1 through 3), and subinstruction DV4 has nine actions (4 through 12).

32-156. Instruction DV E divides the fractional double-precision quantity contained in registers A and L by the fractional single-precision quantity stored at location E of E Memory (or in a CP register). The quantity in E must not include an overflow bit. The absolute value of the fractional quantity contained in (A, L) must always be smaller than the absolute value of the fractional quantity contained in E. (This implies that A cannot contain an overflow bit.) The operation DV E with $0024 \leq E \leq 1776$ can be formulated as follows:

- (1) Set $c(A) = b(A, L) + c(E)$, signs in A and L need not agree, L must not contain an overflow bit.
Set $c(L) = \text{remainder}$.
If $c(E) = 00000$ or 77777 , $c(A) = 037777$ or 140000 , respectively.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction DV E, and j being the instruction stored at location I+1.
Set $c(S) = \text{relevant address of } j$.
Set $c(SQ) = \text{order code of } j$.
- (3) Set $c(Z) = b(Z+1) = I+2$.
- (4) Restore $c(E) = b(E)$ and $c(I+1) = b(I+1)$ if E and/or (I+1) represent an address in E Memory.

Point (2) implies that instruction j is executed next.

32-157. Special Cases of DV E:

- a. DV A divides $b(A, L)$ by $b(A)$ but the sign is reversed if $b(A)$ is positive. DV L is not possible.
- b. DV Q and DV Z can be used if Q and Z do not contain a positive or negative overflow bit.
- c. DV EBANK, DV FBANK, and DV BBANK could be used but are not very useful.
- d. DV ZERO results in 037777 or 140000.
- e. Instructions DV E with $0010 \leq E \leq 0017$ follow the rules of paragraph 32-156.
- f. Instructions DV E with $0020 \leq E \leq 0023$ also follow the rules of paragraph 32-156 (the content of E is not edited when being restored).

32-158. Principle of Operation

32-159. A division as performed by instruction DV E is carried out in ONE's complement arithmetic and in a way similar to a division done manually with binary numbers. Assume first that registers A and L, and location E, contain the positive quantities indicated in figure 32-32. The dividend consists of 28 bits stored in registers A and L, the divisor of 14 bits stored in location E. Because the value of the dividend is smaller than that of the divisor, the division can be started by writing down the first 15 bits of the dividend. Since the 15 bit number is larger than the divisor, the divisor can be subtracted and a ONE can be written into the quotient immediately following the binary point. The division can then be continued conventionally, except that binary, instead of decimal numbers are used. To prove the correctness, the division has been carried out in octal numbers and the quotient has been multiplied by the divisor to result in the dividend.

32-160. In figure 32-32, the same division is carried out but in a way resembling the operation of instruction DV E. Instead of subtracting the divisor from partial remainders, the divisors are added to the complemented remainders. Thus, the subtractions are carried out in inverted form. As a starting step, the first 15 value bits of the dividend (contained in registers A and L), are complemented and written down. Then the divisor is added to the first number in ONE's complement arithmetic which is the same as subtracting the divisor from the recomplemented form of the first number. The ONE's complement sum is a negative number indicating that the divisor has been successfully subtracted; the first remainder is expressed in ONE's complement form and a ONE can be written as the first bit of the quotient.

After adding the next complemented bit of the dividend to the remainder, the divisor is added again. This time the sum is positive indicating that the divisor is too large for subtraction, therefore a ZERO must be entered into the quotient. For the next addition, the previous remainder, plus the added bit, plus the next complemented bit of the dividend, must be used.

32-161. The division can be continued as shown in the figure. Whenever an inverse subtraction is successful, a ONE must be written into the quotient and the next complemented bit of the dividend has to be added to the remainder. Whenever a subtraction is unsuccessful, a ZERO must be written into the quotient and the next complemented bit of the dividend has to be added to that remainder which resulted from the last successful subtraction together with bits which had been added already.

32-162. Actual Execution

32-163. When instruction DV E is executed, the three actions of subinstruction DV0 (row 28 of table 32-4) and the first nine actions (4 through 12) of subinstruction DV1 (row 29) prepare the registers for the actual division. The last three actions (1 through 3) of subinstruction DV1, all actions of subinstructions DV3, DV7, and DV6 (rows 30 through 32) and the first two actions (4 and 5) of subinstruction DV4 (row 33) perform the actual division. The remaining actions (6 through 12) of subinstruction DV6 place the results in the proper registers.

32-164. Actions 1 through 3 of subinstruction DV0 and actions 4 through 12 of subinstruction DV1 do the following:

- a. Establish sign agreement of $c(L)$ with $c(A)$.
- b. Complement $c(A)$ and enter the complemented quantity into B if $c(A, L)$ represents a positive number. Enter $c(A)$ into B if $c(A, L)$ represents a negative number. Thus, a negative quantity, (the high order part of the dividend) is always entered into B; or zero is entered into B if A contains zero.
- c. Shift $c(L)$ one place to the left and enter the shifted quantity into L if $c(A, L)$ represents a positive number. Complement $c(L)$ and shift it one place to the left, and enter the complemented shifted quantity into L if $c(A, L)$ represents a negative number. Thus, a positive quantity (low order part of dividend) is always entered into L; or zero is entered into L if applicable.
- d. Enter $c(E)$ into A if $c(E)$ is a positive quantity. Complement $c(E)$ and enter the complemented quantity into A if $c(E)$ is a negative quantity. Thus, a positive divisor is always entered into A.

- e. A ONE is entered into bit position 16 of register Z if $c(A, L)$ represents a negative number. A ONE is entered into bit position 15 of register Z if $c(E)$ represents a negative number. Thus, the quotients will be positive if bits 16 and 15 are identical; or will be negative if the two bits are not identical.

32-165. When the signs in A and L do not agree, the sign agreement can be established by adding 177776 (minus 1) to $c(A)$ and adding 040000 to $c(L)$, or by adding 000001 to $c(A)$ and adding 137777 to $c(L)$.

For instance $c(A, L) = (012346, 173377) = (012345, 033400)$
 or $c(A, L) = (165431, 004400) = (165432, 144377)$

The method by which instruction DV E establishes sign agreement is based on the same mathematical principle but is implemented in a different way.

32-166. If $c(A, L)$ represents a positive quantity, the quantity 040000 is always added to $c(L)$. If L did contain a positive quantity, a ONE is entered into bit position 15 by the addition; however, the content of the other bit positions is left unchanged. When $c(L)$ is read out, bit position 16 is read into WA's 16 and 15, thus eliminating the ONE in bit position 15. If L did contain a negative quantity (without overflow bit), the ONE added to the ONE contained in bit position 15 changes it to a ZERO, changes the sign bit to a ZERO, and causes end around carry. For instance, adding 040000 to 173377 results in 033400. When $c(L)$ is read out, bit 15 is lost again, but this does not change the quantity.

32-167. If $c(A, L)$ represents a negative number, the quantity 040000 is added to the complemented $c(L)$ to provide the same end effect.

32-168. Figure 32-34 describes how actions 1 through 3 of subinstruction DV0 and actions 4 through 12 of subinstruction DV1 set the registers. First the original content (a) of register A is entered into register B and tested for sign and minus zero. If quantity (a) is positive, it is complemented and again entered into register B; quantity (a) is then tested for plus zero. If it is not equal to zero, the sign originally stored in register A defines the sign of $c(A, L)$. If the quantity (a) is equal to zero, the sign originally contained in register L defines the sign of $c(A, L)$.

32-169. Once the sign of (a, l), the quantity originally contained in registers A and L, has been defined, the operation branches in one of two directions.

(text continued on page 32-132)

c(A) = 012345

c(L) = 033400

c(E) = 21212

$$\frac{.0101001110010111011100000000}{.10001010001010} = .10011010111001$$

```

.0101001110010111011100000000
010100111001011
10001010001010
-----
00011101000001101
  10001010001010
  010111100000111
  10001010001010
  -----
0011000111110110
  10001010001010
  -----
0011110110110000
  10001010001010
  011011001001100
  10001010001010
  -----
010011110000100
  10001010001010
  -----
00010011111010000
  10001010001010
  -----
00010101000110

```

c(A) = 023271

c(L) = 002506

Proof

$$.2471356000 \div .42424 = .46562$$

```

.2471356000
212120
-----
350156
317170
-----
307660
254544
-----
331140
317170
-----
117500
105050
-----
012430

```

$$.46562 \times .42424$$

```

.46562
232710
-----
115344
232710
-----
115344
232710
-----
2471343350
  012430
-----
2471.356000

```

Figure 32-32. Principle of Division, Manual Method

c(A) = 012345
 c(A) = 01010011100101

c(L) = 033400
 c(L) = 11011100000000

c(E) = 021212
 c(E) = 10001010001010

```

    1101011000110100
    0010001010001010
  1  1111000101111100
     0010001010001010
  0  0001010000000111
     1110001011111001
     0010001010001010
  0  0000010110000100
     1100010111110010
     0010001010001010
  1  X1101000011111000
     0010001010001010
  1  X1110011100000100
     0010001010001010
  0  0000100110001111
     1100111000001001
     0010001010001010
  1  X1110000100100111
     0010001010001010
     0000001110110010
  0  1100001001001111
     0010001010001010
  1  X1100100110110011
     1010001010001010
  1  X1101100001111011
     0010001010001010
  1  X1111011000001011
     0010001010001010
  0  0001100010010110
     1110110000010111
     0010001010001010
  0  0000111010100010
     1101100000101111
     0010001010001010
  1  1111101010111001
  Remainder      000010101000110
    
```

Quotient = 10011010111001
 c(A) = 023271

1's indicate successful
 subtractions
 0's indicate unsuccessful
 subtractions

c(L) = 002506

Figure 32-33. Principle of Division, Machine Method

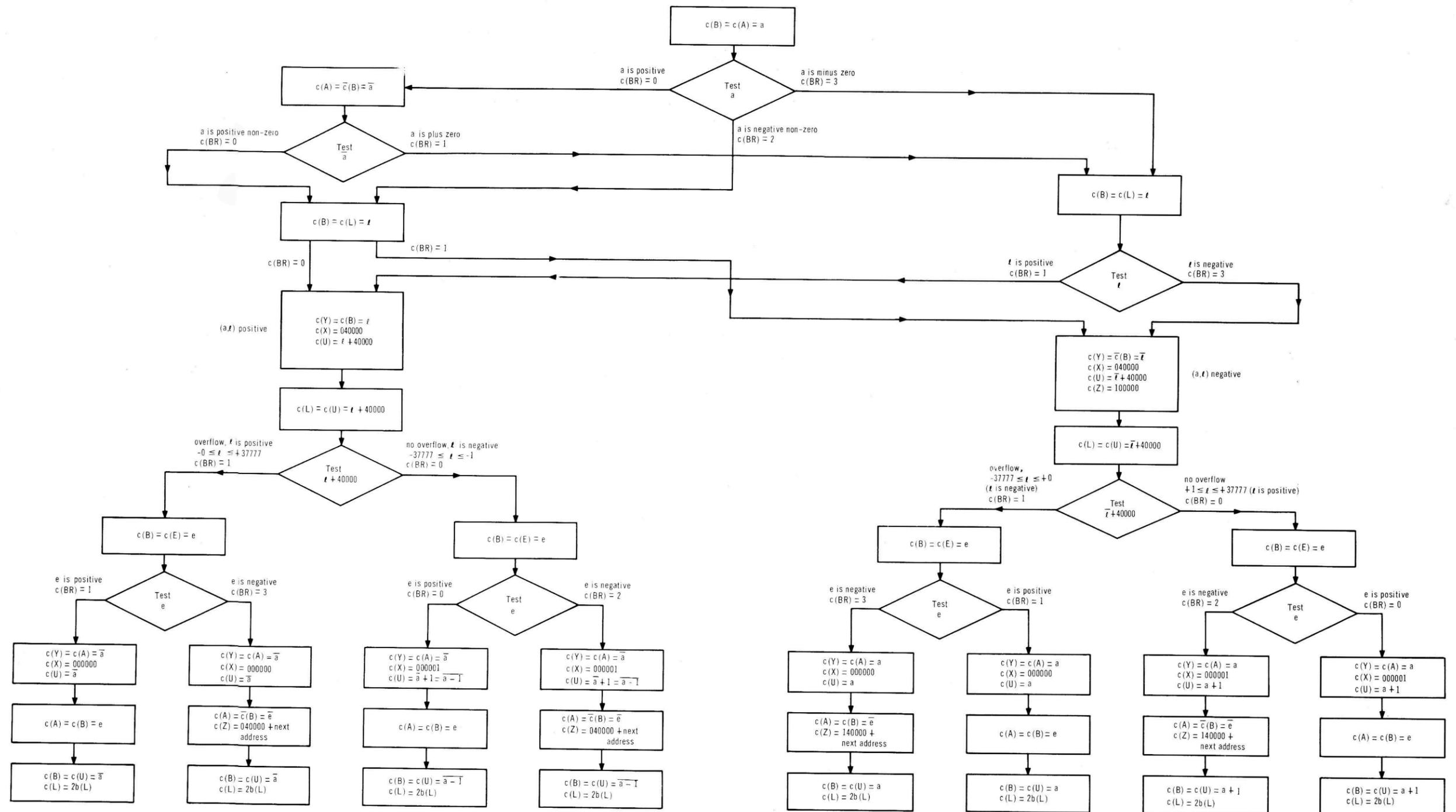
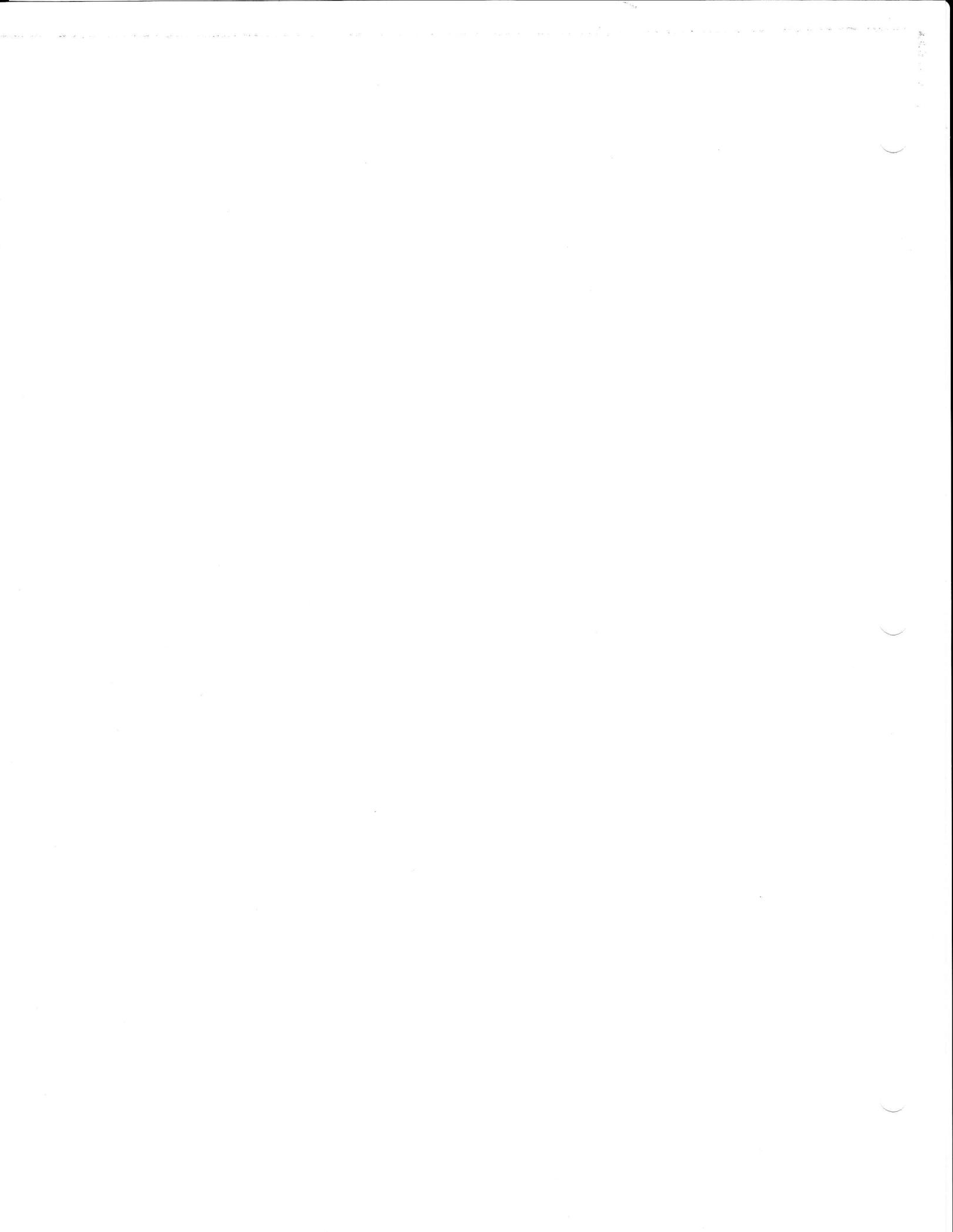


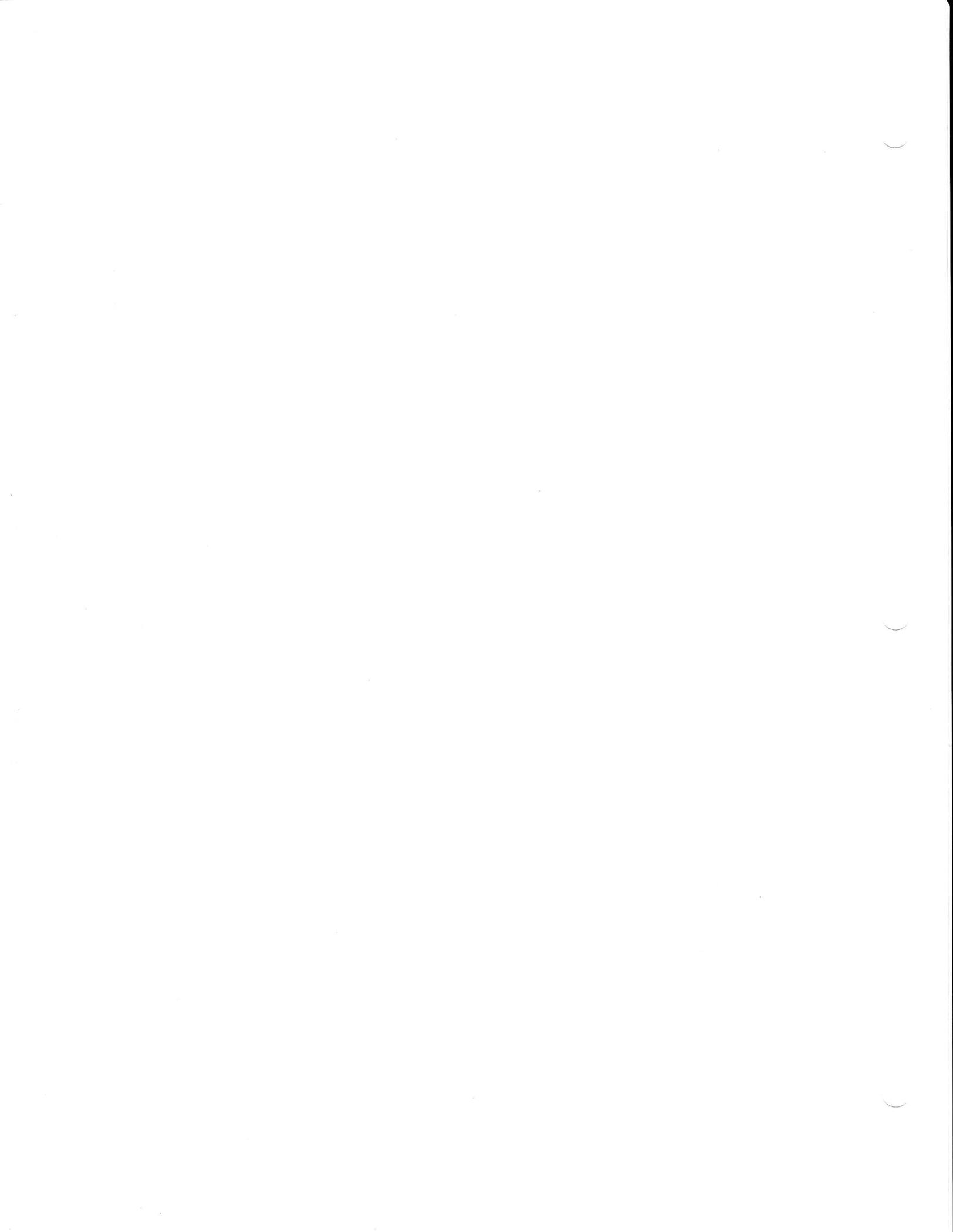
Figure 32-34. Divide Instruction, Flow Diagram (Sheet 1 of 2)



Examples

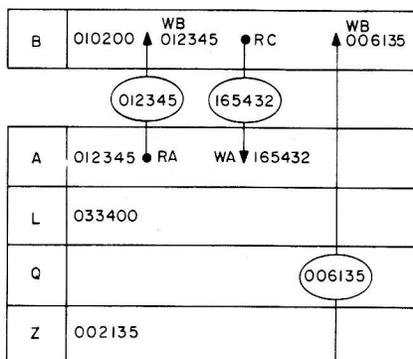
- | | | | | | | | |
|--|---|--|--|---|---|--|--|
| <p>1. a = 012345
 ℓ = 033400
 e = 021212</p> <p>c(B) = 165432
 c(L) = 067000
 c(A) = 021212
 c(Z) = next address</p> | <p>2. a = 012345
 ℓ = 033400
 e = 156565</p> <p>c(B) = 165432
 c(L) = 067000
 c(A) = 021212
 c(Z) = 040000 +
 next address</p> | <p>3. a = 012346
 ℓ = 173377
 e = 021212</p> <p>c(B) = 165432
 c(L) = 067000
 c(A) = 021212
 c(Z) = next address</p> | <p>4. a = 012346
 ℓ = 173377
 e = 156565</p> <p>c(B) = 165432
 c(L) = 067000
 c(A) = 021212
 c(Z) = 040000 +
 next address</p> | <p>5. a = 165432
 ℓ = 144377
 e = 156565</p> <p>c(B) = 165432
 c(L) = 067000
 c(A) = 021212
 c(Z) = 140000 +
 next address</p> | <p>6. a = 165432
 ℓ = 144377
 e = 021212</p> <p>c(B) = 165432
 c(L) = 067000
 c(A) = 021212
 c(Z) = 100000 +
 next address</p> | <p>7. a = 165431
 ℓ = 004400
 e = 156565</p> <p>c(B) = 165432
 c(L) = 067000
 c(A) = 021212
 c(Z) = 140000 +
 next address</p> | <p>8. a = 165431
 ℓ = 004400
 e = 026212</p> <p>c(B) = 165432
 c(L) = 067000
 c(A) = 021212
 c(Z) = 100000 +
 next address</p> |
| <p>9. a = 000000 or
 177777
 ℓ = 005162
 e = 021212</p> <p>c(B) = 000000 or
 177777
 c(L) = 012344
 c(A) = 021212
 c(Z) = next address</p> | <p>10. a = 000000 or
 177777
 ℓ = 005162
 e = 156565</p> <p>c(B) = 000000 or
 177777
 c(L) = 012344
 c(A) = 021212
 c(Z) = 040000 +
 next address</p> | | | <p>11. a = 000000 or
 177777
 ℓ = 172615
 e = 156565</p> <p>c(B) = 000000 or
 177777
 c(L) = 012344
 c(A) = 021212
 c(Z) = 140000 +
 next address</p> | <p>12. a = 000000 or
 177777
 ℓ = 172615
 e = 021212</p> <p>c(B) = 000000 or
 177777
 c(L) = 012344
 c(A) = 021212
 c(Z) = next address</p> | | |

Figure 32-34. Divide Instruction, Flow Diagram (Sheet 2 of 2)



F	
E	
H	
S	0200

G	010200
---	--------

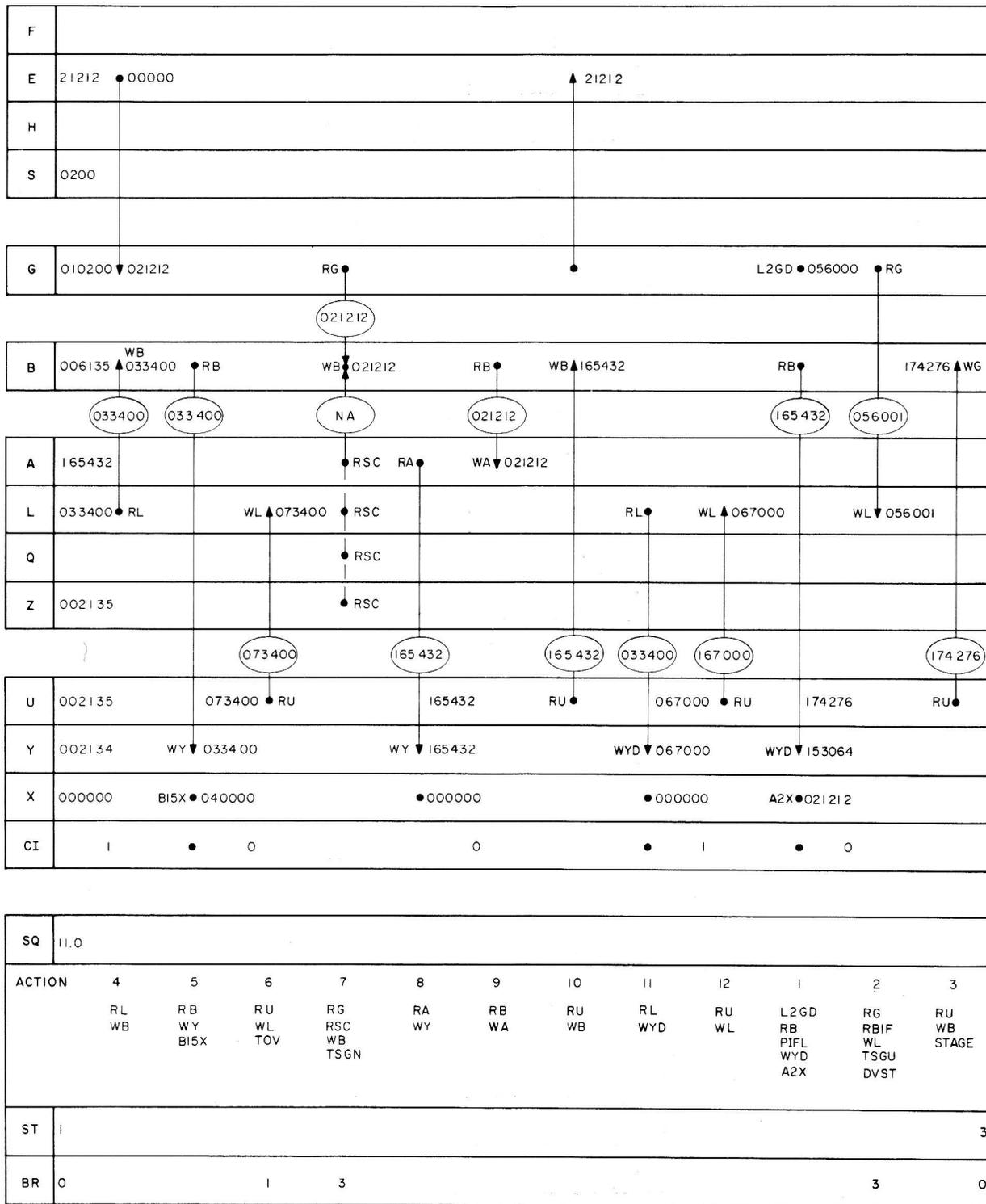


U	002135	RU ●
Y	002134	
X	000000	
CI	I	

SQ	11.0		
ACTION	1	2	3
	RA	RC	RU
	WB	WA	WB
	TSGN	TMZ	STAGE
	TMZ	DVST	
ST	0		I
BR	0	0	0

2770A

Figure 32-35. Subinstruction DVO



2771A

Figure 32-36. Subinstruction DVI.

F	
E	
H	
S	0200

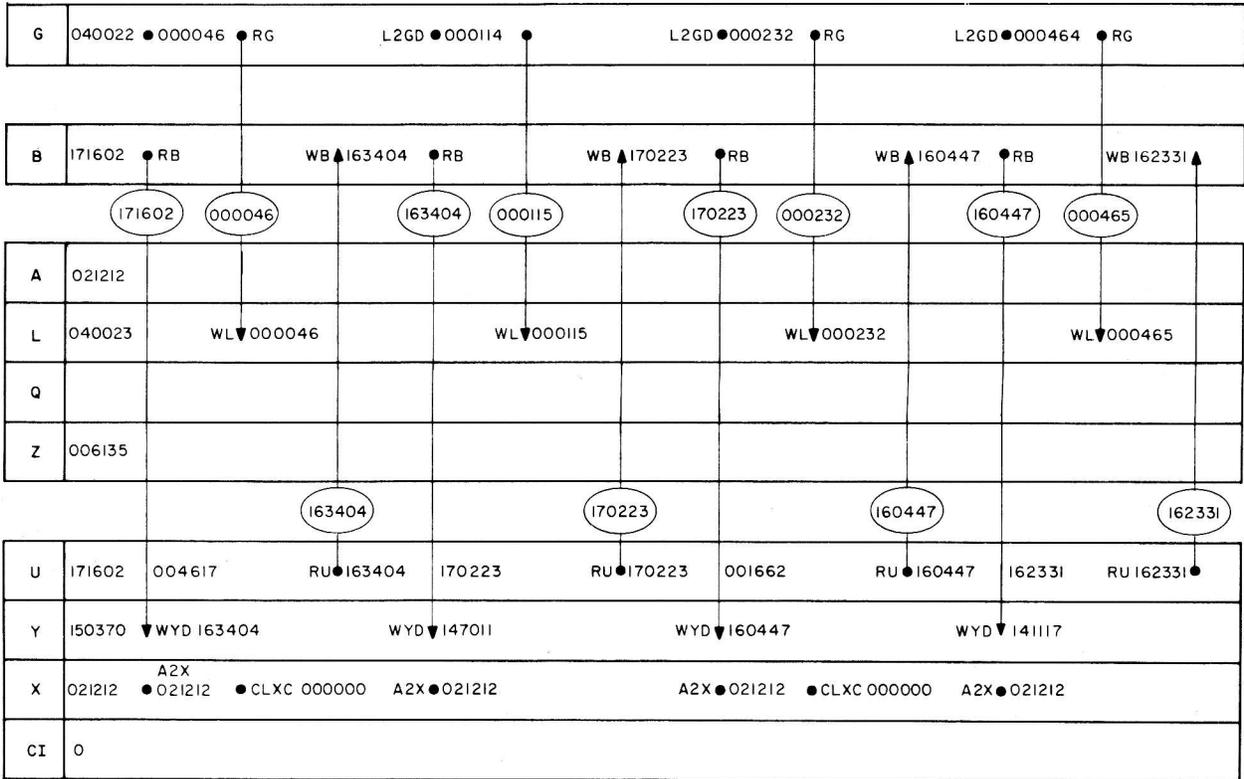
G	L2GD 056000 ● 034002 ● RG		L2GD ● 070004 ● RG		L2GD ● 060010 ● RG		L2GD ● 040022 ● RG	
B	174276 ● RB	WB ▲ 170574 ● RB	WB ▲ 161371 ● RB	WB ▲ 164174 ● RB	171602 ▲ WB			
	(174276)	(034002)	(170574)	(070004)	(161371)	(060011)	(164174)	(040023)
A	021212							
L	056001	WL ▼ 034002	WL ▼ 070004	WL ▼ 060011	WL ▼ 040023			
Q								
Z	006135							
		(170574)	(161371)	(164174)	(171602)			
U	174276	012007	RU ● 170574	002604	RU ● 161371	164174	RU ● 164174	171602
Y	153064	WYD ▼ 170574	WYD ▼ 161371	WYD ▼ 142762	WYD ▼ 150370			
X	021212	A2X ● 021212	CLXC ● 000000	A2X ● 021212	CLXC ● 000000	A2X ● 021212	A2X ● 021212	
CI	0	0	0	0	0	0	0	

SQ	11.0											
TIME	4	5	6	7	8	9	10	11	12	1	2	3
	PIFL FINDS LI5=1	TSGU SETS C(BR)=OX AND CAUSES CLXC		PIFL FINDS LI5=0	TSGU SETS C(BR)=OX AND CAUSES CLXC		PIFL FINDS LI5=1	TSGU SETS C(BR)=IX AND CAUSES RBI WHICH PLACES 000001 ONTO WRITE LINES		PIFL FINDS LI5=1	TSGU SETS C(BR)=IX AND CAUSES RBI WHICH PLACES 000001 ONTO WRITE LINES DVST SETS STAGE COUNTER TO III	STAGE
ST	3											7
BR	0											0

2772A

Figure 32-37. Subinstruction DV3

F	
E	
H	
S	0200



SQ	110											
TIME	4	5	6	7	8	9	10	11	12	1	2	3
	PIFL FINDS LI5=1	TSGU SETS C(BR)=OX AND CAUSES CLXC		PIFL FINDS LI5=0	TSGU SETS C(BR)=IX AND CAUSES RBIF WHICH PLACES 000001 ONTO WRITE LINES		PIFL FINDS LI5=0	TSGU SETS C(BR)=OX AND CAUSES CLXC		PIFL FINDS LI5=0	TSGU SETS C(BR)=IX AND CAUSES RBIF WHICH PLACES 000001 ONTO WRITE LINES - DVST SETS STAGE COUNTER TO 110	STAGE
ST	7											6
BR	0											0

2773A

Figure 32-38. Subinstruction DV7

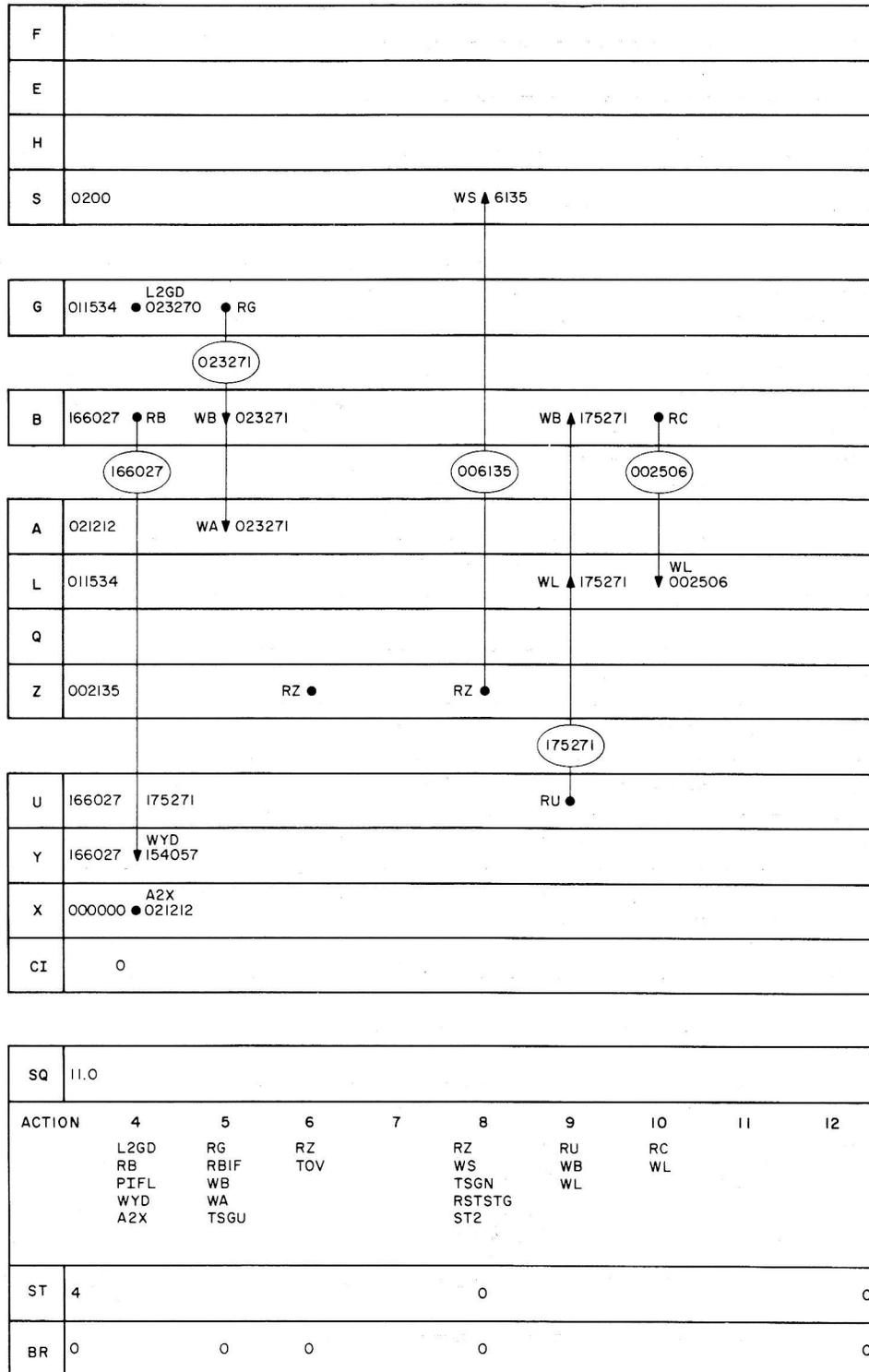
F	
E	
H	
S	0200

G	000464	L2GD ● 001152	● RG	L2GD ● 002326	● RG	L2GD ● 004656	● RG	L2GD ● 011534	● RG
B	162331	● RB	WB ▲ 166075	● RB	WB ▲ 175405	● RB	WB ▲ 173013	● RB	WB 166027 ▲
	(162331)	(001153)	(166075)	(002327)	(173013)	(004656)	(166027)	(001154)	
A	021212								
L	000465	WL ▼ 001153		WL ▼ 002327		WL ▼ 004656		WL ▼ 011534	
Q									
Z	006135								
			(166075)	(175405)	(173013)		(166027)		
U	162331	166075	RU ● 166075	175405	RU ● 175405	014226	RU ● 173013	007242	RU 166027 ●
Y	141117	WYD ▼ 144663	WYD ▼ 154173		WYD ▼ 173013		WYD ▼ 166027		
X	021212	A2X ● 021212	A2X ● 021212		A2X ● 021212	CLXC ● 000000	A2X ● 021212	CLXC ● 000000	
CI	0								

SQ	110											
TIME	4	5	6	7	8	9	10	11	12	1	2	3
	PIFL FINDS LI5=0	TSGU SETS C(BR)=1X AND CAUSES RBIF WHICH PLACES 000001 ON WRITE LINES		PIFL FINDS LI5=0	TSGU SETS C(BR)=1X AND CAUSES RBIF WHICH PLACES 000001 ON WRITE LINES		PIFL FINDS LI5=0	TSGU SETS C(BR)=0X AND CAUSES CLXC		PIFL FINDS LI5=0	TSGU SETS C(BR)=0X AND CAUSES CLXC-DVST SETS STAGE COUNTER TO 110	STAGE
ST	6											4
BR	0											0

2774A

Figure 32-39. Subinstruction DV6



2775A

Figure 32-40. Subinstruction DV4

Initial Conditions:

a, l = 012345; 033400
e = 21212

After DV1-12:

c(B) = 165432
c(L) = 067000
c(A) = 021212

Step 1

DV1-1	L2GD	c(G) = 056000
	RB, WYD, PIFL, L15=1	c(Y) = 153064
	A2X	c(X) = 021212
		c(U) = 174276
DV1-2	RG, WL, set c(BR1) = 1, RB1F	c(L) = 056001
DV1-3	RU, WB	c(B) = 174276

Step 2

DV3-4	L2GD	c(G) = 034002
	RB, WYD, PIFL, L15=1	c(Y) = 170574
	A2X	c(X) = 021212
		c(U) = 012007
DV3-5	RG, WL, set c(BR1) = 0, CLXC	c(U) = 170574
DV3-6	RU, WB	c(L) = 034002
		c(B) = 170574

Step 3

DV3-7	L2GD	c(G) = 070004
	RB, WYD, PIFL, L15=0	c(Y) = 161371
	A2X	c(X) = 021212
		c(U) = 002604
DV3-8	RG, WL, set c(BR1) = 0, CLXC	c(U) = 161371
		c(L) = 070004
DV3-9	RU, WB	c(B) = 161371

Step 4

DV3-10	L2GD	c(G) = 060010
	RB, WYD, PIFL, L15=1	c(Y) = 142762
	A2X	c(X) = 021212
		c(U) = 164174
DV3-11	RG, WL, set c(BR1) = 1, RB1F	c(L) = 060011
DV3-12	RU, WB	c(B) = 164174

Figure 32-41. Actual Division (Sheet 1 of 3)

<u>Step 5</u>		
DV3-1	L2GD	c(G) = 040022
	RB, WYD, PIFL, L15=1	c(Y) = 150370
	A2X	c(X) = 021212
		c(U) = 171602
DV3-2	RG, WL, set c(BR1) = 1, RB1F	c(L) = 040023
DV3-3	RU, WB	c(B) = 171602
<u>Step 6</u>		
DV7-4	L2GD	c(G) = 000046
	RB, WYD, PIFL, L15=1	c(Y) = 163404
	A2X	c(X) = 021212
		c(U) = 004617
DV7-5	RG, WL, set c(BR1) = 0, CLXC	c(U) = 163404
		c(L) = 000046
DV7-6	RU, WB	c(B) = 163404
<u>Step 7</u>		
DV7-7	L2GD	c(G) = 000114
	RB, WYD, PIFL, L15=0	c(Y) = 147011
	A2X	c(X) = 021212
		c(U) = 170223
DV7-8	RG, WL, set c(BR1) = 1, RB1F	c(L) = 000115
DV7-9	RU, WB	c(B) = 170223
<u>Step 8</u>		
DV7-10	L2GD	c(G) = 000232
	RB, WYD, PIFL, L15=0	c(Y) = 160447
	A2X	c(X) = 021212
		c(U) = 001662
DV7-11	RG, WL, set c(BR1) = 0, CLXC	c(U) = 160447
		c(L) = 000232
DV7-12	RU, WB	c(B) = 160447
<u>Step 9</u>		
DV7-1	L2GD	c(G) = 000464
	RB, WYD, PIFL, L15=0	c(Y) = 141117
	A2X	c(X) = 021212
		c(U) = 162331
DV7-2	RG, WL, set c(BR1) = 1, RB1F	c(L) = 000465
DV7-3	RU, WB	c(B) = 162331

Figure 32-41. Actual Division (Sheet 2 of 3)

<u>Step 10</u>		
<u>DV6-4</u>	L2GD	c(G) = 001152
	RB, WYD, PIFL, L15=0	c(Y) = 144663
	A2X	c(X) = 021212
		c(U) = 166075
DV6-5	RG, WL, set c(BR1) = 1, RB1F	c(L) = 001153
DV6-6	RU, WB	c(B) = 166075
<u>Step 11</u>		
<u>DV6-7</u>	L2GD	c(G) = 002326
	RB, WYD, PIFL, L15=0	c(Y) = 154173
	A2X	c(X) = 021212
		c(U) = 175405
DV6-8	RG, WL, set c(BR1) = 1, RB1F	c(L) = 002327
DV6-9	RU, WB	c(B) = 175405
<u>Step 12</u>		
<u>DV6-10</u>	L2GD	c(G) = 004656
	RB, WYD, PIFL, L15=0	c(Y) = 173013
	A2X	c(X) = 021212
		c(U) = 014226
DV6-11	RG, WL, set c(BR1) = 0, CLXC	c(U) = 173013
		c(L) = 004656
DV6-12	RU, WB	c(B) = 173013
<u>Step 13</u>		
<u>DV6-1</u>	L2GD	c(G) = 011534
	RB, WYD, PIFL, L15=0	c(Y) = 166027
	A2X	c(X) = 021212
		c(U) = 007242
DV6-2	RG, WL, set c(BR1) = 0, CLXC	c(U) = 166027
		c(L) = 011534
DV6-3	RU, WB	c(B) = 166027
<u>Step 14</u>		
<u>DV4-4</u>	L2GD	c(G) = 023270
	RB, WYD, PIFL, L15=0	c(Y) = 154057
	A2X	c(X) = 021212
		c(U) = 175271
DV4-5	RG, WB, WA, set c(BR1) = 1, RB1F	c(B)=c(A) = 023271
<u>Final Sequence</u>		
DV4-6	RZ, set c(BR) = 00	
DV4-7	no effect	
DV4-8	RZ, WS, set c(BR1) = 0, c(BR) = 00, c(S) = b(Z)	
DV4-9	RU, WB, WL	c(B)=c(L) = 175271
DV4-10	RC, WL	c(L) = 002506
		c(A) = 023271

Figure 32-41. Actual Division (Sheet 3 of 3)

From there on the flow chart is self explanatory. Examples for the various branches are shown on sheet 2. Figure 32-35 through 32-40 demonstrate the execution of the first example. The quotient is entered into register L bit by bit as the low order part of the dividend is shifted (and complemented) bit by bit into register B via register Y and adder output gates (U). For details refer to figure 32-41.

32-170. Action 7 of subinstruction DV4 complements the quotient if the quotient must be negative. Actions 9 and 10 recompute the remainder if the remainder must be positive. Action 8 enters the address of the next instruction into register S and subinstruction STD2 calls forward the next subinstruction as usual.

32-171. INSTRUCTION ADS E

32-172. Instruction ADS E (Add to Storage E) is a Basic Instruction which is represented by order code 02.6 and a 10 bit address. Instruction ADS E consists of subinstructions ADS0 and STD2, the execution of which takes two MCT's.

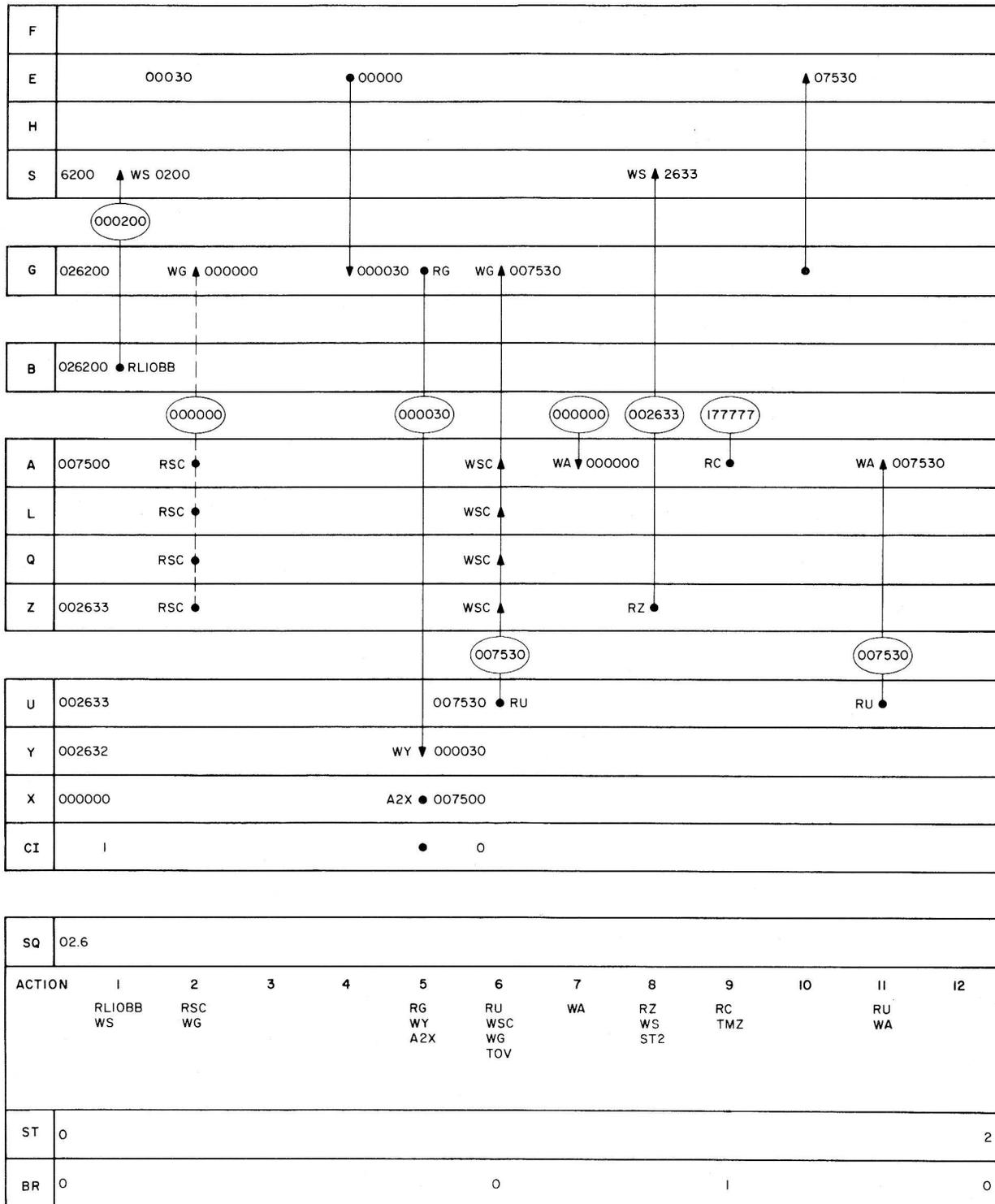
32-173. Instruction ADS E adds the quantity in register A to the quantity in location E of E Memory (or a CP register), stores the sum in A with overflow bit, and stores the sum in E without overflow bit if E represents an address in E Memory. The operation ADS E with $0024 \leq E \leq 1777$ can be formulated as follows:

- (1) Set $c(E) = b(E) + b(A)$ except positive or negative overflow bit.
Set $c(A) = b(E) + b(A)$ with positive or negative overflow bit.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction ADS E,
and j being the instruction stored at location (I+1).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(I+1) = b(I+1)$ if (I+1) represents an address in E
Memory.

Point (2) implies that instruction j is executed next.

32-174. Special Cases of ADS E:

- a. ADS A doubles the content of register A whereby any overflow bit is included in the new content of A.
- b. ADS L, ADS Q, and ADS Z also enter an overflow bit into registers L, Q, and Z, respectively.



2715A

Figure 32-42. Subinstruction ADS0

- c. ADS EBANK, ADS FBANK, and ADS BBANK can be used but the particular read and write operations must be observed.
- d. ADS ZERO has no purpose.
- e. Instructions ADS E with $0010 \leq E \leq 0017$ follow the rules of paragraph 32-173.
- f. Instructions ADS E with $0020 \leq E \leq 0023$ also follow the rules of paragraph 32-173 except that the sum is edited as it is entered into E.

32-175. When instruction ADS E is executed, action 1 of subinstruction ADS0 (row 34 of table 32-4) replaces the quantity contained in register S with the 10 bit address thus erasing the quarter code contained in S. The quantity from location E is entered into register G at time 2 or 4. Action 5 adds the quantities in G and A, and action 6 enters the sum into register G or into another CP register. At time 10, the sum without any overflow bit is entered into an E Memory location if one was addressed. If positive or negative overflow occurred during the addition, 000001 or 177776, respectively, is entered into register A by action 7, however, action 11 replaces this quantity by the sum including an overflow bit. Action 8 enters the address of the next instruction into register S and subinstruction STD2 calls forward the next instruction as usual.

32-176. Figure 32-42 illustrates the execution of subinstruction AD0 of instruction ADS 0200. Initially, location 200 contains quantity 00030 and register A contains quantity 007500.

32-177. INSTRUCTION DAS E

32-178. Instruction DAS E (Double Add to Storage E) is a Basic Instruction which is represented by order code 02.0 and a 10 bit address. Instruction DAS E consists of subinstructions DAS0, DAS1, and STD2, the execution of which takes three MCT's.

32-179. Instruction DAS E adds the double precision quantity contained in registers A and L to the double precision quantity stored at locations E and E+1 of E Memory (or two CP registers) and stores in A the overflow resulting from the addition as a whole. The operation DAS E with $0024 \leq E \leq 1776$ excluding the last address of any E Memory bank (table 32-2) can be formulated as follows:

- (1) Set $c(E, E+1) = b(E, E+1) + c(A, L)$ where $c(E)$ includes any overflow resulting from $b(E+1) + c(L)$ but not any overflow resulting from the addition as a whole.

Set $c(A) = 000000$ if no net overflow occurred.
 Set $c(A) = 000001$ if net positive overflow occurred.
 Set $c(A) = 177776$ if net negative overflow occurred.
 Set $c(L) = 000000$.

- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction DAS E , and j being the instruction stored at location $(I+1)$.
 Set $c(S) =$ relevant address of j .
 Set $c(SQ) =$ order code of j .
- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(I+1) = b(I+1)$ if $(I+1)$ represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-180. If an overflow occurs during the addition of $b(E+1)+c(L)$, $b(A)$ is incremented by one (in case of positive overflow), or decremented by one (negative overflow) before $b(E)$ and $c(A)$ are added. If positive or negative overflow occurs during the second addition, the quantity 000001 (positive overflow) or 177776 (negative overflow) is stored in A , otherwise the quantity 000000 is stored in A . The $c(L)$ is set to 000000. The sum which is stored at E and $E+1$ may contain two different sign bits and 28 value bits.

32-181. Special Cases of DAS E :

- a. DAS A , (alternate code DDOUBL for double precision double) doubles the double precision quantity contained in registers A and L whereby the final content of A includes any overflow bit resulting from the second addition, i. e., $b(A) + [b(A) + \text{overflow of first addition}]$. The $b(A)$ must not include an overflow bit.
- b. DAS 0010 and DCA 0013 (table 30-4) are useful and follow the rules of paragraph 32-182.
- c. Any DAS E with $0000 \leq E \leq 0022$ must be used with extreme care so as not to destroy stored data; E must not be 0023 in order to prevent destruction of data in counter $T2$. Whenever locations 0020 through 0023 are involved, the sum is edited as it is stored.

32-182. When instruction DAS is executed, first the sum $c(L)+c(E+1)$ is computed by subinstruction DAS0 which also stores the sum in location $E+1$, adds 000001 or 177776 to the $b(A)$ if an overflow occurred, and stores this new quantity in the Adder. Thereafter, subinstruction DAS1 computes the sum $c(A)+c(E)$ and stores it in location E .

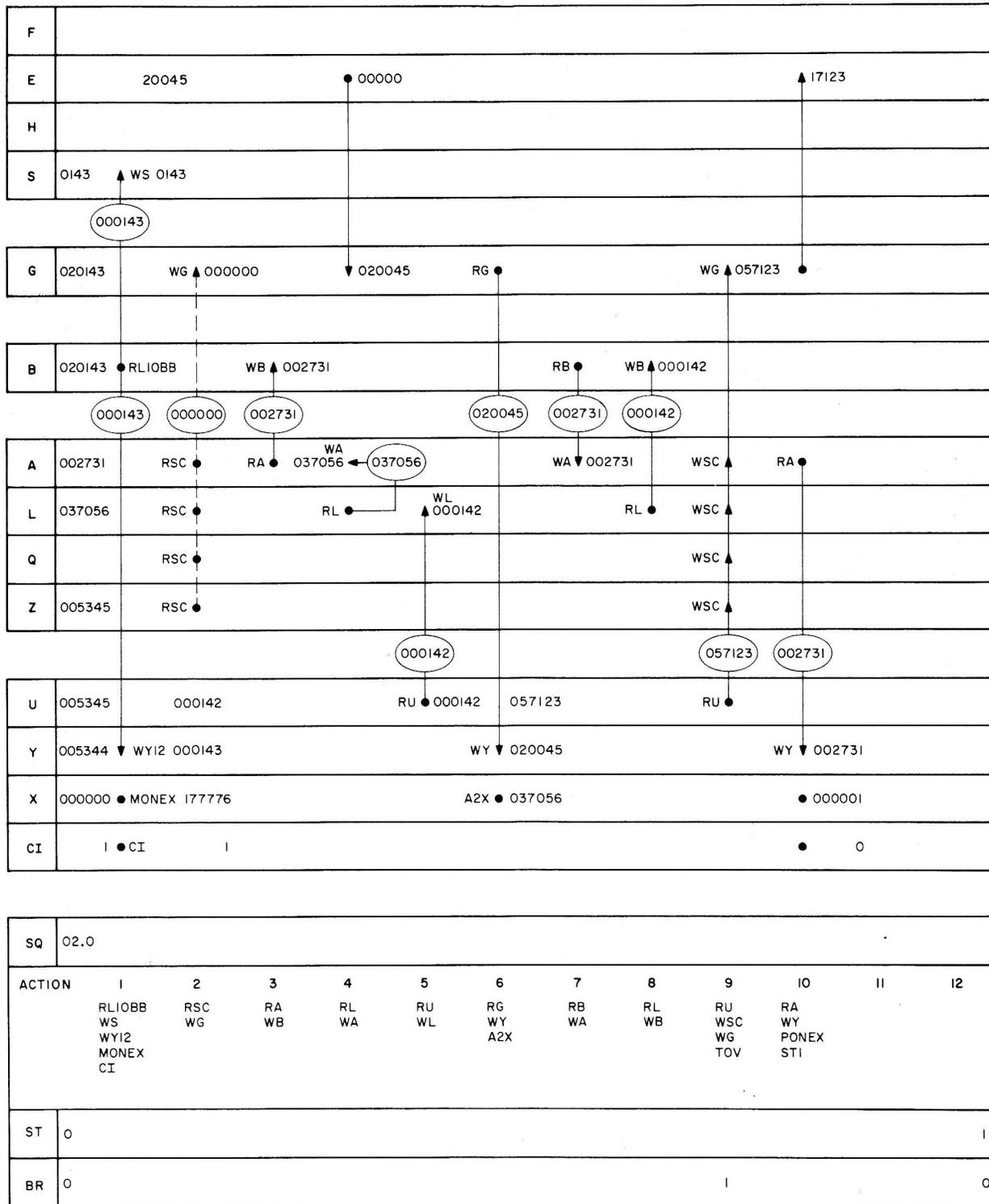
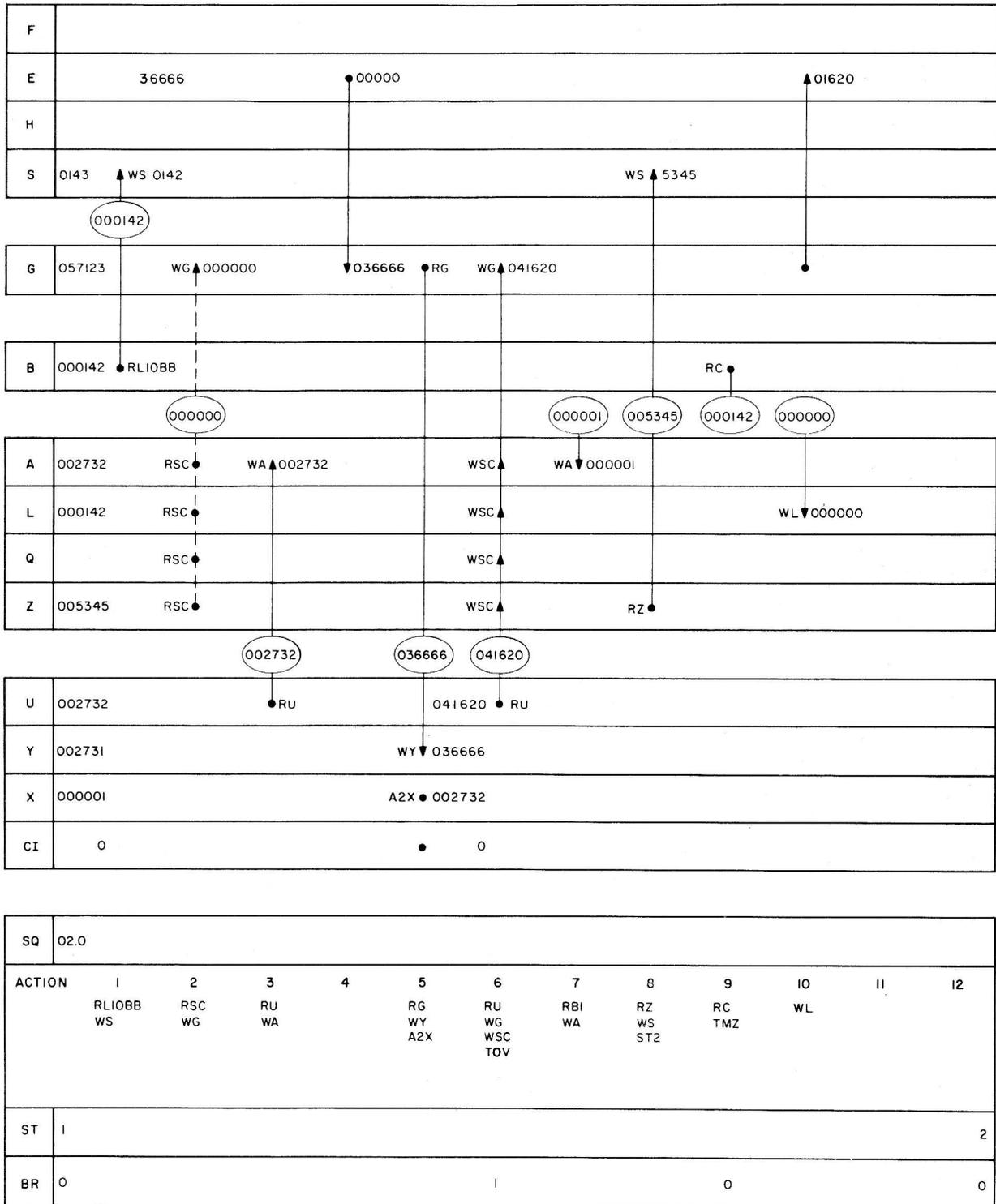


Figure 32-43. Subinstruction DAS0



2708A

Figure 32-44. Subinstruction DAS1

The Yul Programming System accomplishes this by replacing instruction DAS E with code DAS (E+1) which is wired into the program. As the AGC executes subinstruction DAS0, relevant address (E+1) is available first and is decremented by one. Subinstruction DAS1 then uses the decremented address E. For execution of subinstructions DAS0 and DAS1 refer to rows 35 and 36 of table 32-4. When double precision quantities are added to storage, address E must not be equal to the last address of any E Memory bank in order to allow (E+1) to be the next address in the same bank.

32-183. The execution of instruction DAS 0142 is illustrated in figures 32-43 and 32-44. The instruction is stored at location 5344. Location 0142 contains quantity 36666 and location 0143, quantity 20045 to which quantities 002731 and 037056, contained in registers A and L, are to be added. Note, that registers B, G, and S contain relevant address 0143 instead of 0142 at the start of subinstruction DAS0. Thus, E Memory enters quantity 20045 into register G at time 4 and action 6 enters the quantity into the Adder together with quantity 037056 which was temporarily stored in register A. Action 9 transfers the sum 057123 to register G from where it is entered into location (E+1) at time 10. Action 9 also tests the sum, finds a positive overflow bit and sets the branch flip-flops to 01. Because of this, the quantity 000001 is added to the original content of register A. (If no overflow had occurred, the quantity 000000 would have been added; if negative overflow had occurred, the quantity 177776 would have been added.)

32-184. Action 1 of subinstruction DAS0 decrements address (E+1) to obtain address E = 0142 which is temporarily stored in registers L and B before it is entered into register S by action 1 of subinstruction DAS1. The quantity 36666 is entered into register G at time 4 and entered into the Adder by action 5 together with the incremented quantity of A. The final sum is transferred to location E via register G. Because a positive overflow occurred during the second addition, action 7 enters the quantity 000001 into register A. (Otherwise, quantity 000000 or 177776 would have been entered.) Action 8 enters the address of the next instruction into register S and subinstruction STD2 calls forward the next instruction as usual.

32-185. If address E has been 0000 (A), the first sum, 057123, would have been entered into register L by action 9 of DAS0 and action 10 of DAS1 would not have entered 000000 into L. Furthermore, action 11 of DAS1 would have replaced the quantity 000001 in A by the second sum 041620.

32-186. INSTRUCTION INCR E

32-187. Instruction INCR E (Increment E) is a Basic Instruction which is represented by order code 02.4 and a 10 bit address.

Instruction INCR E consists of subinstructions INCR0 and STD2, the execution of which takes two MCT's.

32-188. Instruction INCR E increments by one the quantity stored at location E in E Memory (or a CP register). The operation INCR E with $0024 \leq E \leq 1777$ can be formulated as follows:

- (1) Set $c(E) = b(E)+1$ except overflow bit which is lost.

If overflow occurs when a certain counter is addressed, one of the following operations is requested by the Counter Priority Control:

<u>Counter Addressed</u>		<u>Operation</u>
0025	T1	Instruction PINC 0024 or PINC T2 is executed. $\triangle 1$
0026	T3	Instruction RUPT and RUPT Transfer Routine 3 are executed. $\triangle 2$
0027	T4	Instruction RUPT and RUPT Transfer Routine 4 are executed. $\triangle 2$
0030	T5	Instruction RUPT and RUPT Transfer Routine 5 are executed. $\triangle 2$

$\triangle 1$ Refer to table 30-4, EMA's 0024 and 0025.

$\triangle 2$ Refer to tables 30-4 and 30-6.

- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction INCR E, and j being the instruction stored at location (I+1).
 Set $c(S) =$ relevant address of j.
 Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(I+1) = b(I+1)$ if (I+1) represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-189. Special Cases of INCR E:

- a. INCR A, INCR L, INCR Q, and INCR Z are useful. An incremented quantity entered into A, L, Q, or Z may also contain an overflow bit.

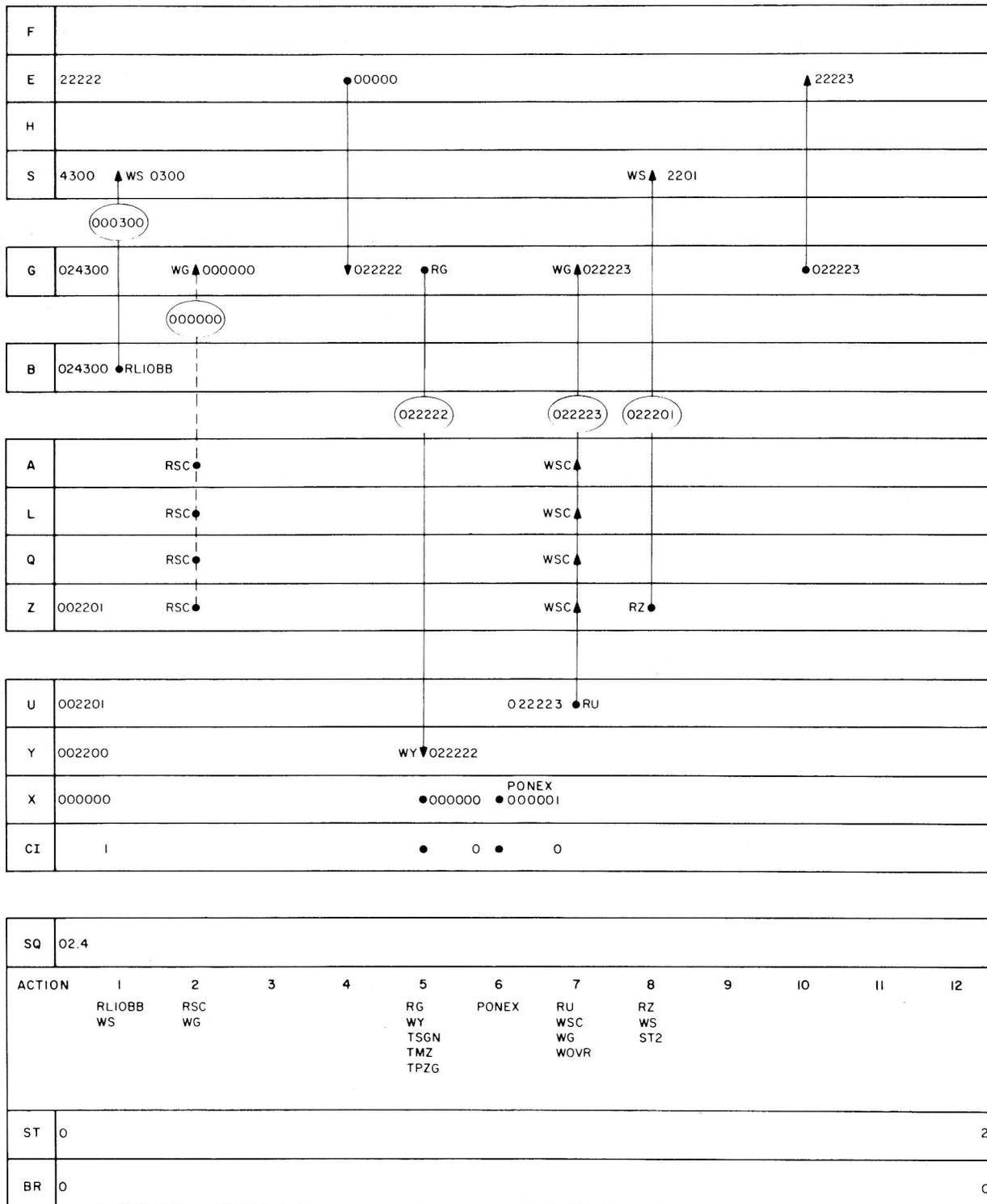


Figure 32-45. Subinstruction INCR0

- b. INCR EBANK and INCR FBANK have no purpose. INCR BBANK can be used to increment the content of register EBANK.
- c. INCR ZERO has no purpose.
- d. Instructions INCR E with $0010 \leq E \leq 0017$ follow the rules of paragraph 32-188.
- e. Instructions INCR E with $0020 \leq E \leq 0023$ also follow the rules of paragraph 32-188 except that the incremented quantity is edited as it is entered into location E.

32-190. When instruction INCR E is executed, action 1 of subinstruction INCR0 (row 37 of table 32-4) replaces the quantity contained in register S by the 10 bit address thus erasing the quarter code contained in S. The quantity from location E is entered into register G at time 2 or 4 and entered into the Adder by action 5. Action 6 adds the quantity 000001 to the content of the Adder. Action 7 enters the sum into register G from where it is transferred to location E in E Memory at time 10. If a CP register is addressed, the incremented quantity is entered by action 7. Action 7 also tests the incremented quantity for overflow and signals the Counter Priority Control if a certain counter is addressed. Action 8 enters the address of the next instruction into register S and subinstruction STD2 calls forward the next instruction as usual.

32-191. Figure 32-45 illustrates the execution of subinstruction INCR0 of instruction INCR 0300. Location 0300 initially contains quantity 22222.

32-192. INSTRUCTION AUG E

32-193. Instruction AUG E (Augment E) is an Extra Code Instruction which is represented by order code 12.4 and a 10 bit address. Instruction AUG E must be preceded by Special Instruction EXTEND which enters a ONE into bit position EXT of register SQ. Instruction AUG E consists of subinstructions AUG0 and STD2, the execution of which takes two MCT's.

32-194. Instruction AUG E increases by one the absolute value of the quantity stored at location E in E Memory (or a CP register). The operation AUG E with $0024 \leq E \leq 1777$ can be formulated as follows:

- (1) If $c(E)$ is positive, set $c(E) = b(E)+1$, except overflow bit which is lost.
If $c(E)$ is negative, set $c(E) = b(E)-1$, except overflow bit which is lost.

If overflow occurs when a certain counter is addressed, one of the following operations is requested by the Counter Priority Control:

<u>Counter Addressed</u>		<u>Operation</u>
0025	T1	Instruction PINC 0024 or PINC T2 is executed. $\triangle 1$
0026	T3	Instruction RUPT and RUPT Transfer Routine 3 are executed. $\triangle 2$
0027	T4	Instruction RUPT and RUPT Transfer Routine 4 are executed. $\triangle 2$
0030	T5	Instruction RUPT and RUPT Transfer Routine 5 are executed. $\triangle 2$

$\triangle 1$ Refer to table 30-4, EMA's 0024 and 0025

$\triangle 2$ Refer to tables 30-4 and 30-6.

- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction AUG E, and j being the instruction stored at location (I+1).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(I+1) = b(I+1)$ if (I+1) represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-195. Special Cases of AUG E

- a. AUG A, AUG L, AUG Q, and AUG Z are useful. An augmented quantity entered into A, L, Q, or Z may also contain an overflow bit.
- b. AUG EBANK and AUG FBANK have no purpose. AUG BBANK can be used to increment or decrement the content of register EBANK if bit 16 of register FBANK is known.
- c. AUG ZERO has no purpose.
- d. Instructions AUG E with $0010 \leq E \leq 0017$ follow the rules of paragraph 32-194.

- e. Instructions AUG E with $0024 \leq E \leq 0023$ also follow the rules of paragraph 32-194 except that the augmented quantity is edited as it is entered into location E.

32-196. The execution of instruction AUG E is similar to that of instruction INCR E. Action 6 of subinstruction INCR0 (row 37 of table 32-4) always adds the quantity 000001 (plus one) to the content of location E. Action 6 of subinstruction AUG0 (row 38) adds the quantity 000001 to the content of E only if location E contains a positive quantity and enters 177776 (minus one) if E contains a negative quantity.

32-197. INSTRUCTION DIM E

32-198. Instruction DIM E (Diminish E) is an Extra Code Instruction which is represented by order code 12.6 and a 10 bit address. Instruction DIM E must be preceded by Special Instruction EXTEND which enters a ONE into bit position EXT of register SQ. Instruction DIM E consists of subinstructions DIM0 and STD2, the execution of which takes two MCT's.

32-199. Instruction DIM E decreases by one the absolute value of the quantity stored at location E in E Memory (or a CP register). The operation DIM E with $0024 \leq E \leq 1777$ can be formulated as follows:

- (1) If $c(E)$ is positive nonzero, set $c(E) = b(E) - 1$.
If $c(E)$ is negative nonzero, set $c(E) = b(E) + 1$.
If $c(E)$ is plus or minus zero, set $c(E) = b(E)$.
- (2) Set $c(B) = b(I+1) = j$, I being the address of instruction DIM E, and j being the instruction stored at location (I+1).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z) + 1 = I + 2$.
- (4) Restore $c(I+1) = b(I+1)$ if (I+1) represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-200. Special Cases of DIM E:

- a. DIM A, DIM L, DIM Q, and DIM Z are useful.
- b. DIM EBANK and DIM FBANK have no purpose. DIM BBANK can be used to decrement or increment the content of register EBANK if the content of bit position 16 of register FBANK is known.
- c. DIM ZERO has no purpose.

- d. Instructions DIM E with $0010 \leq E \leq 0017$ follow the rules of paragraph 32-199.
- e. Instructions DIM E with $0020 \leq E \leq 0023$ also follow the rules of paragraph 32-199 except that the augmented quantity is edited as it is entered into location E.

32-201. The execution of instruction DIM E is similar to that of instructions INCR E and AUG E. Action 6 of subinstruction DIM0 (row 39 of table 32-4) adds the quantity 177776 to the content of location E if E contains a positive nonzero quantity, adds the quantity 000001 if E contains a negative nonzero quantity, and adds the quantity 000000 if E contains plus or minus zero.

32-202. INSTRUCTION MSU E

32-203. Instruction MSU E (Modular Subtract E) is an Extra Code Instruction which is represented by order code 12.0 and a 10 bit address. Instruction MSU E must be preceded by Special Instruction EXTEND which enters a ONE into bit position EXT of register SQ. Instruction MSU E consists of subinstructions MSU0 and STD2, the execution of which takes two MCT's.

32-204. Instruction MSU computes the ONE's complement difference from the cyclic TWO's complement numbers stored in register A and location E. The operation MSU E with $0024 \leq K \leq 1777$ can be formulated as follows:

- (1) Set $c(A) = b'(A) - c'(E)$ where $b'(A)$ and $c'(E)$ are cyclic TWO's complement numbers and $c(A)$ is a ONE's complement number.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction MSU E and j being the instruction stored at location (I+1).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z) + 1 = I + 2$.
- (4) Restore $c(E) = b(E)$ and $c(I+1) = b(I+1)$ if E and/or (I+1) represent an address in E Memory.

Point (2) implies that instruction j is executed next.

32-205. Special Cases of MSU E

- a. MSU A enters 000000 into A whether bits 16 and 15 of $b(A)$ agree or not.
- b. Register L, Q, Z, EBANK and FBANK normally do not contain cyclic TWO's complement numbers.

- c. MSU ZERO does not change the content of register A if bit position 16 of A contains a ZERO but decrements by one the c(A) if bit position 16 of A contains a ONE.
- d. Locations 0010 through 0024 normally do not contain cyclic TWO's complement numbers.

32-206. Many navigational computations require the calculation of the difference of two angles. Angular information stored in several counters (table 30-4) is expressed in cyclic TWO's complement numbers. Instruction MSU E is provided to calculate the difference of two cyclic TWO's complement numbers and to supply the angular difference in a ONE's complement number for use in further calculations.

32-207. Cyclic TWO's complement numbers contained in a memory location E indicate angular quantities as shown below.

c(E)	angle	c(A)
00000	0°	000000
10000	45°	010000
20000	90°	020000
30000	135°	030000
40000	180° = -180°	140000
50000	225° = -135°	150000
60000	270° = -90°	160000
70000	315° = -45°	170000
00000	360° = 0°	000000

When these quantities are transferred from a location E to register A, the quantities shown in the third column appear in A.

32-208. Before instruction MSU E is executed, one cyclic TWO's complement number, the minuend, is entered into register A. When instruction MSU E is executed, the subtrahend is transferred from location E to register G at time 4 of subinstruction MSU E (row 40 of table 30-4). Action 6 enters the minuend into register X of the Adder, the complemented subtrahend into register Y, and forces a carry bit into bit position one, thus performing the addition in TWO's complement arithmetic (as during instruction MP). Action 7 enters bit 15 (normally the overflow bit) provided by the output gates into bit positions 16 and 15 of register A while the other bits provided by the Adder are entered into the corresponding bit positions of register A.

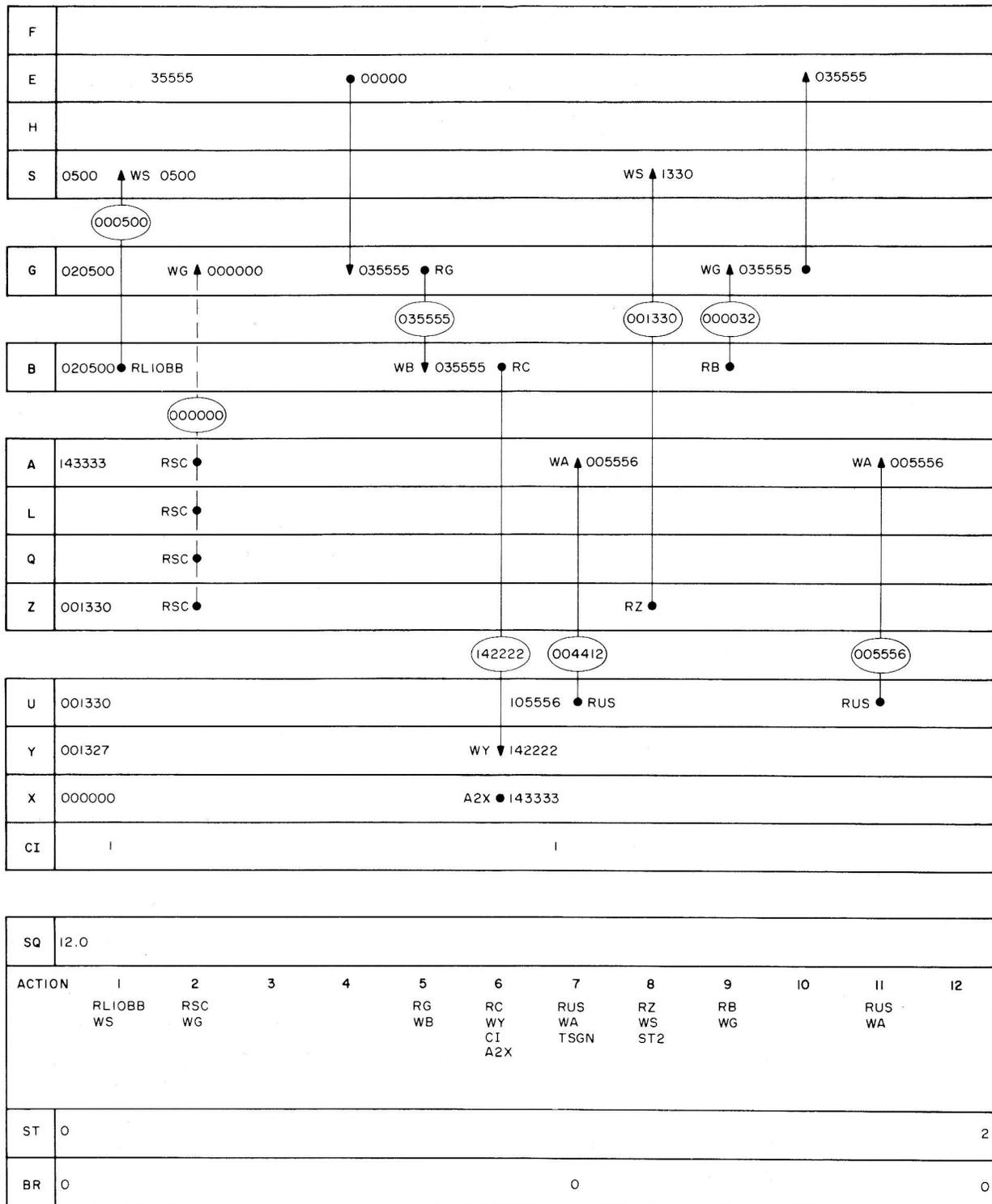


Figure 32-46. Subinstruction MSU0

32-209. Action 7 also tests bit 15 provided by the Adder. If bit 15 is a ZERO, indicating that the difference angle is positive (smaller than 180°) no action is taken at time 10 and the $c(A)$ is the final angle difference. If bit 15 is a ONE, indicating that the difference angle is negative (180° or larger), the quantity one is subtracted from the $c(A)$ to convert the cyclic TWO's complement number to a ONE's complement number. Action 9 returns the subtrahend to register G for restoring in E Memory and action 8 enters the address of the next instruction into register S. Subinstruction STD2 calls forward the next instruction as usual.

32-210. Figure 32-46 illustrates the execution of subinstruction MSU0 of instruction MSU 0500. The minuend is 43333, $c(A) = 143333$, and the subtrahend is 35555. The remainder is 05556.

32-211. Further examples are given to demonstrate various operational conditions.

- a. Assume $c(E1) = 30000$ (135°) and $c(E2) = 20000$ (90°). By transferring $c(E1)$ to register A and executing MSU E2, the following computation is performed:

$$\begin{aligned} c(Y) &= \overline{c(A)} = 030000 \\ c(X) &= \overline{c(E2)} = 157777 \\ CI &= \underline{\quad 1} \\ c(U) &= \underline{010000} \text{ no carry around because of control pulse CI} \\ \text{final } c(A) &= 010000 \text{ (} 45^\circ \text{)} \\ c(E3) &= 010000 \text{ if the result is transferred to location E3.} \end{aligned}$$

- b. Assume $c(E1) = 70000$ (315°) and $c(E2) = 60000$ (270°) and the same operation is performed.

$$\begin{aligned} c(Y) &= \overline{c(A)} = 170000 \\ c(X) &= \overline{c(E)} = 017777 \\ CI &= \underline{\quad 1} \\ c(U) &= 010000 \\ \text{final } c(A) &= 010000 \text{ (} 45^\circ \text{)} \\ c(E3) &= 010000 \end{aligned}$$

- c. Assume $c(E1) = 50000$ (225°) and $c(E2) = 30000$ (135°)

$$\begin{aligned} c(Y) &= \overline{c(A)} = 150000 \\ c(X) &= \overline{c(E2)} = 147777 \\ CI &= \underline{\quad 1} \\ c(U) &= 120000 \\ \text{final } c(A) &= 020000 \text{ } c(U15) \text{ is entered into A16 and A15 as in} \\ &\text{all examples} \\ c(E3) &= 020000 \end{aligned}$$

d. Assume $c(E1) = 10000 (45^\circ)$ and $c(E2) = 70000 (315^\circ)$.

$$\begin{aligned} c(Y) = c(A) &= 010000 \\ c(X) = \overline{c}(E2) &= 017777 \\ CI &= \underline{\quad 1} \\ c(U) &= 020000 \\ \text{final } c(A) &= 020000 \\ c(E3) &= 020000 \end{aligned}$$

e. Assume $c(E1) = 20000 (90^\circ)$ and $c(E2) = 30000 (135^\circ)$.

$$\begin{aligned} c(Y) = c(A) &= 020000 \\ c(X) = \overline{c}(E2) &= 147777 \\ CI &= \underline{\quad 1} \\ c(U) &= 170000 \\ c(A) &= 170000 \\ \text{plus } &\underline{177776} \text{ because a ONE was entered into A16} \\ c(U) &= 167777 \\ \text{final } c(A) &= 167777 \\ c(E3) &= 67777 = -10000 (-45^\circ) \end{aligned}$$

f. Assume $c(E1) = 60000 (270^\circ)$ and $c(E2) = 70000 (315^\circ)$.

$$\begin{aligned} c(Y) = c(A) &= 160000 \\ c(X) = \overline{c}(E2) &= 007777 \\ CI &= \underline{\quad 1} \\ c(U) &= 170000 \\ c(A) &= 170000 \\ \text{plus } &\underline{177776} \\ c(U) &= 167777 \\ \text{final } c(A) &= 167777 \\ c(E3) &= 67777 = -10000 (-45^\circ) \end{aligned}$$

g. Assume $c(E1) = 30000 (135^\circ)$ and $c(E1) = 30000 (135^\circ)$ and $c(E2) = 50000 (225^\circ)$

$$\begin{aligned} c(Y) = c(A) &= 030000 \\ c(X) = \overline{c}(E2) &= 027777 \\ CI &= \underline{\quad 1} \\ c(U) &= 060000 \\ c(A) &= 160000 \\ \text{plus } &\underline{177776} \\ c(U) &= 157777 \\ \text{final } c(A) &= 157777 \\ c(E3) &= 57777 = -20000 (-90^\circ) \end{aligned}$$

h. Assume $c(E1) = 70000$ (315°) and $c(E2) = 10000$ (45°).

$$\begin{array}{rcl}
 c(Y) = c(A) & = & 170000 \\
 c(X) = \bar{c}(E2) & = & 167777 \\
 CI & = & \underline{1} \\
 c(U) & = & 160000 \\
 c(A) & = & 160000 \\
 \text{plus} & & \underline{177776} \\
 c(U) & = & 157777 \\
 \text{final } c(A) & = & 157777 \\
 c(E3) & = & 57777 = -20000 (-90^\circ)
 \end{array}$$

32-212. INSTRUCTION MSK K

32-213. Instruction MSK K (Mask with K) is a Basic Instruction which is represented by order code 07. and a 12 bit address. The alternate spelling of MSK K is MASK K. Instruction MSK K consists of subinstructions MSK0 and STD2, the execution of which takes two MCT's.

32-214. Instruction MSK performs the Boolean operation AND (symbol \wedge) with the content of register A and the data stored at location K. The truth table for each bit position of A and K is shown below.

A	K	$A \wedge K$
0	0	0
0	1	0
1	0	0
1	1	1

The operation MSK K with $0024 \leq K \leq 7777$ can be formulated as follows:

- (1) Set $c(A) = b(A) \wedge c(K)$ whereby bit 15 of $c(K)$ is AND'd with bits 16 and 15 of $c(A)$.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction MSK K, and j being the instruction stored at location (I+1).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(K) = b(K)$ and $c(I+1) = b(I+1)$ if K and/or (I+1) represent an address in E Memory.

32-215. Special Cases of MSK K.

- a. MSK A has no purpose.
- b. MSK L, MSK Q, and MSK Z also AND the overflow bits.

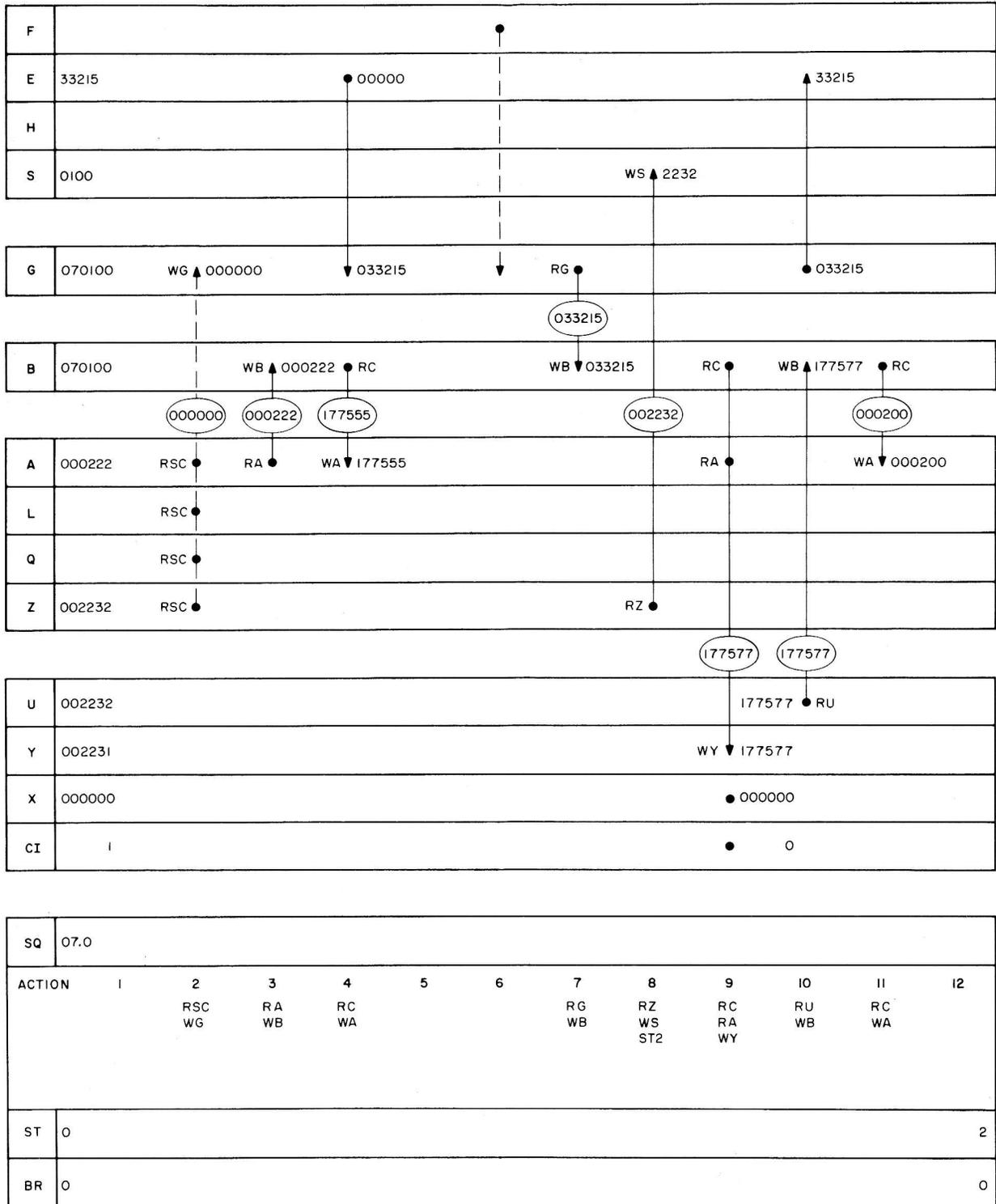


Figure 32-47. Subinstruction MSK0

- c. MSK ZERO sets $c(A) = 000000$.
- d. Instructions MSK K with $0010 \leq K \leq 0023$ follow the rules of paragraph 32-17. (The content of K is not edited when being restored.)

32-216. Instruction MSK K in reality performs the Boolean OR operation $\overline{b(A)} \vee \overline{c(K)}$ instead of the AND operation $b(A) \wedge c(K)$, both having the same effect. When instruction MSK K is executed, the quantity from location K is entered into register G at time 2, 4, or 6 of subinstruction MSK0 (row 41 of table 32-4), and into register B by action 7. Actions 3 and 4 complement the content of register A. Action 9 enters the complemented content of register B (control pulse RC) and the complemented content of register A (control pulse RA) onto the WA's and into register Y. The quantity $\overline{b(A)} \vee \overline{c(K)}$ is provided by the output gates of the Adder. Actions 10 and 11 complement this quantity, enter the complemented quantity, which is the final result $\overline{b(A)} \vee \overline{c(K)} = b(A) \wedge c(K)$, into register A. Action 8 enters the address of the next instruction into register Z and subinstruction STD2 calls forward the next instruction as usual.

32-217. Figure 32-47 illustrates the execution of subinstruction MSK0 of instruction MSK 0100. The quantities AND'd are 000222 and 033215.

32-218. CHANNEL INSTRUCTIONS

32-219. INSTRUCTION READ H

32-220. Instruction READ H (Read H) is a Channel Instruction which is represented by order code 10.0 and a 9 bit channel address (table 30-5). Instruction READ H must be preceded by Special Instruction EXTEND which enters a ONE into bit position EXT of register SQ. Instruction READ H consists of subinstructions READ0 and STD2, the execution of which takes two MCT's.

32-221. Instruction READ H enters the content of channel H into register A. The operation READ H with $005 \leq H \leq 033$ can be formulated as follows:

- (1) Set $c(A) = c(H)$ whereby bit H15 is entered into bit positions A16 and A15.
Retain $c(H)$.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction READ H and j being the instruction stored at location (I+1).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.

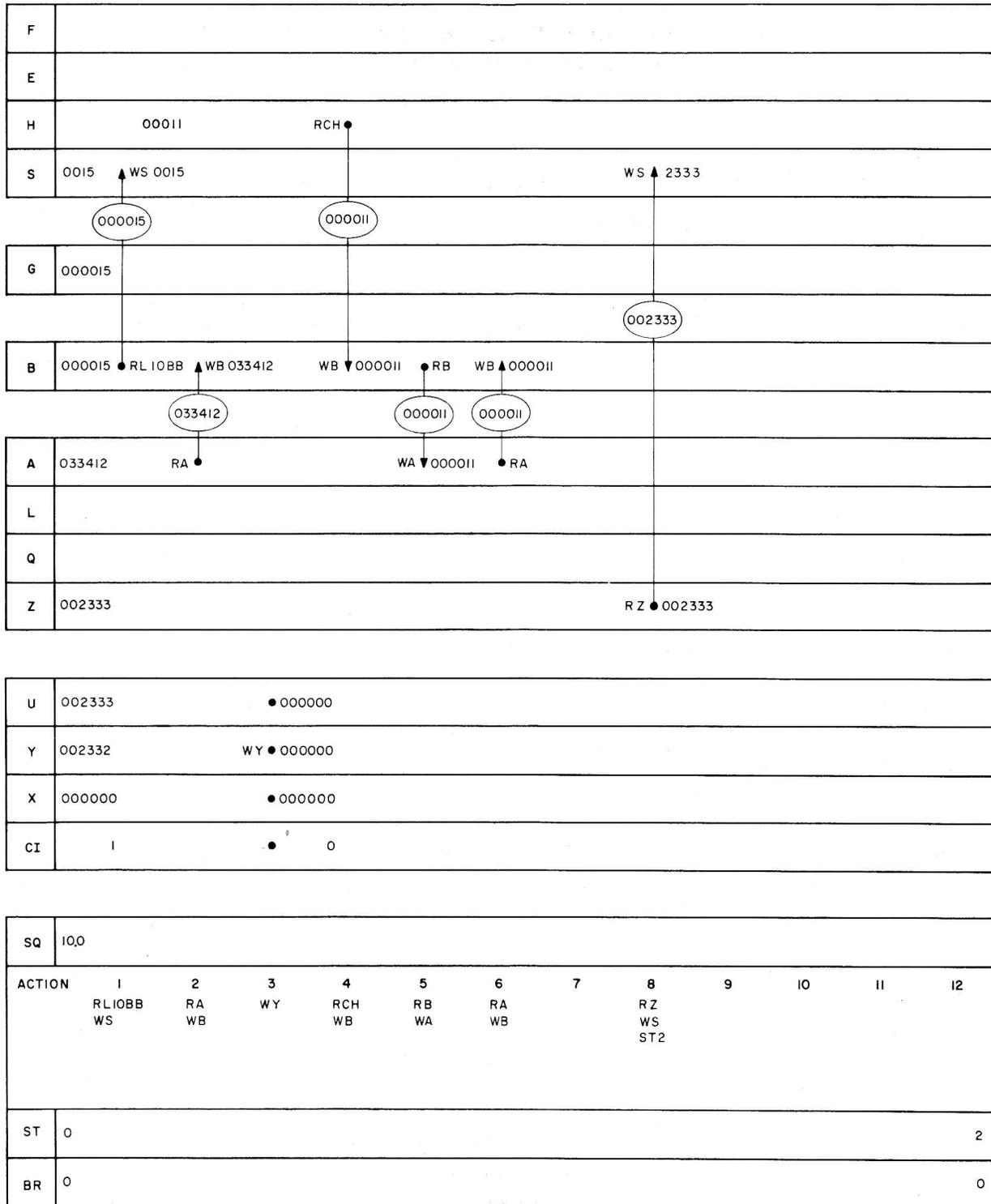


Figure 32-48. Subinstruction READ0

- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(I+1) = b(I+1)$ if $(I+1)$ represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-222. Special Cases of READ H:

- a. READ L and READ Q enter all sixteen bits of $c(L)$ or $c(Q)$ into A.
- b. READ 003 and READ 004 enter fourteen bits of $c(SCALER2)$ or $c(SCALER1)$ into A.
- c. Instructions READ H with $005 \leq H \leq 033$ follow the rules of paragraph 32-221.
- d. Channels 034 and 035 (downlink channels) cannot be read by a Channel Instruction, therefore 000000 is entered into A.

32-223. When instruction READ H is executed, the quantity from channel H is entered into register B by action 4 of subinstruction READ0 (row 42 of table 30-4), and action 5 transfers the quantity to register A. Action 8 enters the address of the next instruction into register S and subinstruction STD2 calls forward the next instruction as usual.

32-224. Figure 32-48 illustrates the execution of subinstruction READ0 of instruction READ 015, channel 15 containing the quantity 00011, a keycode from the keyboard of the main panel DSKY.

32-225. INSTRUCTION WRITE H

32-226. Instruction WRITE H (Write H) is a Channel Instruction which is represented by order code 10.1 and a 9 bit channel address (table 30-5). Instruction WRITE H must be preceded by Special Instruction EXTEND which enters a ONE into bit position EXT of register SQ. Instruction WRITE H consists of subinstructions WRITE0 and STD2, the execution of which takes two MCT's.

32-227. Instruction WRITE H enters the content of register A into channel H. The operation WRITE H with $005 \leq H \leq 014$ can be formulated as follows:

- (1) Set $c(H) = c(A)$ whereby bit A15 is entered into bit position H15 and bit A15 is not transferred.
Keep $c(A)$.

- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction WRITE H, and j being the instruction stored at location $(I+1)$.
Set $c(S) =$ relevant address of j .
Set $c(SQ) =$ order code of j .
- (3) Set $c(Z) = b(Z) + 1 = I+2$.
- (4) Restore $c(I+1) = b(I+1)$ if $(I+1)$ represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-228. Special Cases of WRITE H

- a. WRITE L and WRITE Q enter all sixteen bits of $c(A)$ into L or Q.
- b. SCALER2 and SCALER1 cannot be written into by a Channel Instruction.
- c. Instruction WRITE H with $005 \leq H \leq 014$ follow the rules of paragraph 32-327.
- d. Channels 15 through 33 cannot be written into by a Channel Instruction.
- e. WRITE 034 and WRITE 035 enter bits A16, A14 through A1, and a parity bit into channel 034 or 035.

32-229. The execution of instruction WRITE H is similar to that of instruction READ H. (Compare rows 42 and 43 of table 30-4.) Action 5 of subinstruction READ0 transfers the channel information from register B to register A. Action 5 of subinstruction WRITE0 transfers the content of register A to the addressed channel.

32-230. INSTRUCTION RAND H

32-231. Instruction RAND H (Read and AND H) is a Channel Instruction which is represented by order code 10.2 and a 9 bit channel address (table 30-5). Instruction RAND H must be preceded by Special Instruction EXTEND which enters a ONE into bit position EXT of register SQ. Instruction RAND H consists of subinstructions RAND0 and STD2, the execution of which takes two MCT's.

32-232. Instruction RAND H performs the Boolean operation AND (symbol \wedge) with the contents of register A and channel H and stores the logical product in A. The truth table for each bit of $c(A)$ and $c(H)$ is shown below.

A	H	$A \wedge H$
0	0	0
0	1	0
1	0	0
1	1	1

The operation RAND H with $005 \leq H \leq 033$ can be formulated as follows:

- (1) Set $c(A) = b(A) \wedge c(H)$ whereby bits A16 and A15 are AND'd with bit H15.
Retain $c(H)$.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction RAND H, j being the instruction stored at location (I+1).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(I+1) = b(I+1)$ if (I+1) represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-233. Special Cases of RAND H:

- a. RAND L and RAND Q make use of all sixteen bits in L or Q to form the logical product.
- b. RAND 003 and RAND 004 can be used to form a logical product with the fourteen bits in SCALER2 or SCALER 1.
- c. Instructions RAND H with $005 \leq H \leq 033$ follow the rules of paragraph 32-236.
- d. RAND 034 and RAND 035 cannot form a logical product because downlink channels cannot be read by a Channel Instruction; 000000 is entered into A.

32-234. Instruction RAND H in reality performs $\overline{b(A)} \vee \overline{c(H)} = m$; $c(A) = \overline{m}$ which supplies the same result. When instruction RAND H is executed, action 1 of subinstruction RAND0 (row 44 of table 32-4) replaces the quantity in register S by a 10 bit address, thus erasing two bits of the eighth code. Actions 2 and 3 complement the $b(A)$ and enter the complemented quantity into the Adder which supplies the $\overline{b(A)}$ at its output gates (U). Action 4 enters $c(H)$ into register B. Action 5 reads $\overline{c(H)}$ from the complement side of register B and $\overline{b(A)}$ from the output gates both onto the WA's and into register A thus forming $\overline{b(A)} \vee \overline{c(H)}$. Actions 6 and 7 complement the content of A to provide the desired logical product.

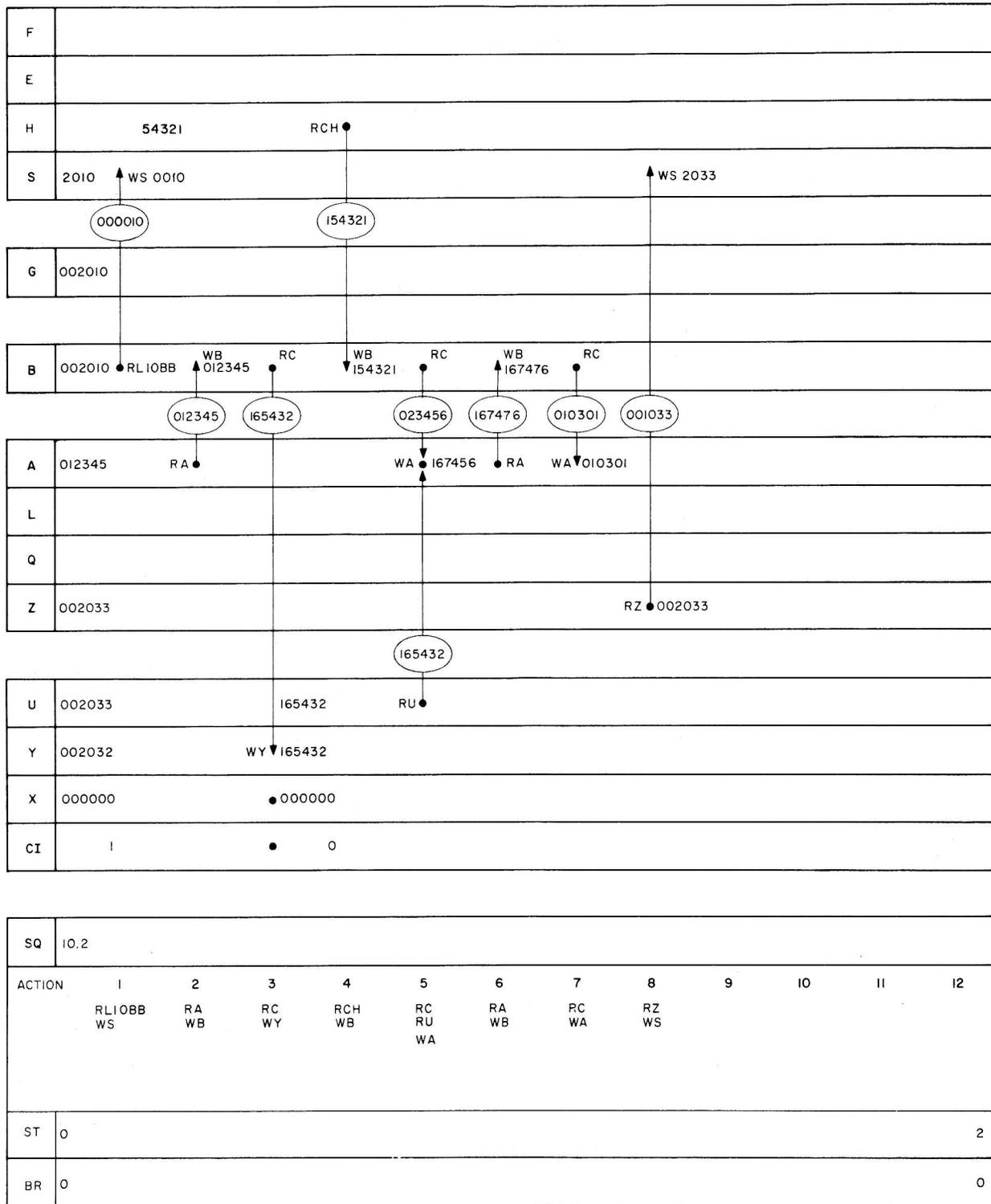


Figure 32-49. Subinstruction RANDO

Action 8 enters the address of the next instruction into register S and sub-instruction STD2 calls forward the next instruction as usual.

32-235. Figure 32-49 illustrates the execution of subinstruction RAND0 of instruction RAND 010, AND'ing quantities 012345 and 154321, the logical product being 010301.

32-236. INSTRUCTION WAND H

32-237. Instruction WAND H (Write and AND H) is a Channel Instruction which is represented by order code 10.3 and a 9 bit channel address (table 30-5). Instruction WAND H must be preceded by Special Instruction EXTEND which enters a ONE into bit position EXT of register SQ. Instruction WAND H consists of subinstructions WAND0 and STD2, the execution of which takes two MCT's.

32-238. Instruction WAND H forms the logical product described in paragraph 32-232 and stores it in register A and in channel H. The operation WAND H with $005 \leq H \leq 014$ can be formulated as follows:

- (1) Set $c(H) = c(A) = b(A) \wedge b(H)$ whereby bits A16 and A15 are AND'd with bit H15.
Retain $c(H)$.
- (2) Set $c(B) = c(I+1) = j$, j being the address of instruction WAND H, and j being the instruction stored at location $(I+1)$.
Set $c(S) =$ relevant address of j .
Set $c(SQ) =$ order code of j .
- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(I+1) = b(I+1)$ if $(I+1)$ represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-239. Special Cases of WAND H:

- a. WAND L and WAND Q make use of all sixteen bits in L or Q to form the logical product.
- b. WAND 003 and WAND 004 cannot enter the logical product into SCALER2 or SCALER1.
- c. Instructions WAND H with $005 \leq H \leq 014$ follow the rules of paragraph 32-238.
- d. Instructions WAND H with $015 \leq H \leq 033$ cannot enter the logical product into these channels.

- e. WAND 034 and WAND 035 cannot form a logical product because downlink channels cannot be read by a Channel Instruction; 000000 is entered into A and H.

32-240. The execution of instruction WAND H is similar to that of instruction RAND H. (Compare rows 44 and 45 of table 32-4.) Action 7 of subinstruction WAND0 also enters the logical product into the addressed channel.

32-241. INSTRUCTION ROR H

32-242. Instruction ROR H (Read and OR H) is a Channel Instruction which is represented by order code 10.4 and a 9 bit channel address (table 30-5). Instruction ROR H must be preceded by Special Instruction EXTEND which enters a ONE into bit position EXT of register SQ. Instruction ROR H consists of subinstructions ROR0 and STD2, the execution of which takes two MCT's.

32-243. Instruction ROR H performs the Boolean operation OR (symbol \vee) with the contents of register A and channel H, and stores the logical sum in A. The truth table for each bit of $c(A)$ and $c(H)$ is shown below.

A	H	$A \vee H$
0	0	0
0	1	1
1	0	1
1	1	1

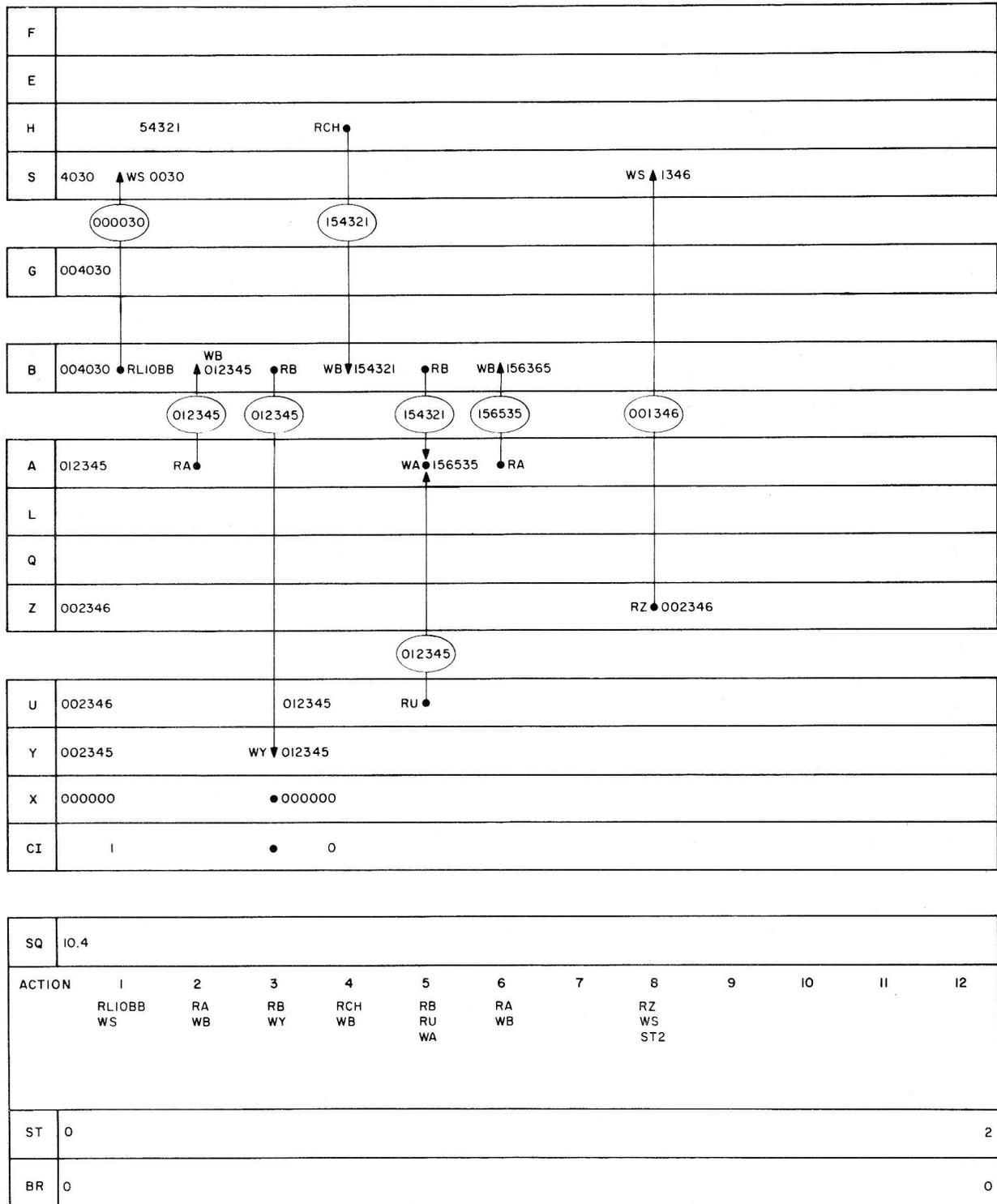
The operation ROR H with $005 \leq H \leq 033$ can be formulated as follows:

- (1) Set $c(A) = b(A) \vee c(H)$ whereby bits A16 and A15 are OR'd with bit H15.
Retain $c(H)$.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction ROR H, and j being the instruction stored at location (I+1).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(I+1) = b(I+1)$ if (I+1) represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-244. Special Cases of ROR H:

- a. ROR L and ROR Q make use of all sixteen bits in L or Q to form the logical sum.



2834A

Figure 32-50. Subinstruction ROR0

- b. ROR 003 and ROR 004 can be used to form a logical sum with the fourteen bits in SCALER2 or SCALER1.
- c. Instructions ROR H with $005 \leq H \leq 033$ follow the rules of paragraph 32-243.
- d. ROR 034 and ROR 035 cannot form a logical sum because downlink channels cannot be read by a Channel Instruction; $b(A)$ is retained in A.

32-245. When instruction ROR H is executed, action 1 of subinstruction ROR0 (row 45 of table 32-4) replaces the quantity in register S by a 10 bit address plus erasing two bits of the eighth code. Actions 2 and 3 enter the content of register A into the Adder which supplies the $c(A)$ at its output gates (U). Action 4 enters $c(H)$ into register B. Action 5 reads $b(A)$ from the Adder and $c(H)$ from register B into the WA's and into register A, thus forming $b(A) \vee c(H)$. Action 8 enters the address of the next instruction into register S and subinstruction STD2 calls forward the next instruction as usual.

32-246. Figure 32-50 illustrates the execution of subinstruction ROR0 of instruction ROR 030, OR'ing quantities 012345 and 154321, the logical sum being 156365.

32-247. INSTRUCTION WOR H

32-248. Instruction WOR (Write and OR H) is a Channel Instruction which is represented by order code 10.5 and a 9 bit channel address (table 30-5). Instruction WOR H must be preceded by Special Instruction EXTEND which enters a ONE into bit position EXT of register SQ. Instruction WOR H consists of subinstructions WOR0 and STD2, the execution of which takes two MCT's.

32-249. Instruction WOR forms the logical sum described in paragraph 32-243 and stores it in register A and in channel H. The operation WOR H with $005 \leq H \leq 013$ can be formulated as follows:

- (1) Set $c(H) = c(A) = b(A) \vee b(H)$ whereby bits A16 and A15 are OR'd with bit H15.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction WOR H and j being the instruction stored at location (I+1).
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = b(Z)+1 = I+2$.
- (4) Restore $c(I+1) = b(I+1)$ if (I+1) represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-250. Special Cases of WOR H:

- a. WOR L and WOR Q make use of all sixteen bits in L or Q to form the logical sum.
- b. WOR 003 and WOR 004 cannot enter the logical sum into SCALER2 or SCALER1.
- c. Instructions WOR H with $015 \leq H \leq 014$ follow the rules of paragraph 32-249.
- d. Instructions WOR H with $015 \leq H \leq 033$ cannot enter the logical sum into these channels.
- e. ROR 034 and ROR 035 cannot form a logical sum because down-link channels cannot be read by a Channel Instruction, b(A) is entered into A and H.

32-251. The execution of instruction WOR H is similar to that of instruction ROR H. (Compare rows 46 and 47 of table 32-4.) Action 5 of subinstruction WOR0 enters the logical sum into the addressed channel also.

32-252. INSTRUCTION RXOR H

32-253. Instruction RXOR H (Read and Exclusive OR H) is a Channel Instruction which is represented by order code 10.6 and a 9 bit channel address (table 30-5). Instruction RXOR H must be preceded by Special Instruction EXTEND which enters a ONE into bit position EXT of register SQ. Instruction RXOR H consists of subinstructions RXOR0 and STD2, the execution of which takes two MCT's.

32-254. Instruction RXOR H performs the Boolean Operation Exclusive OR (symbol \vee) with the contents of register A and channel H, and stores the logical result in A. The truth table for each bit of c(A) and c(H) is shown below.

A	H	$A \vee H$
0	0	0
0	1	1
1	0	1
1	1	0

The operation RXOR H with $005 \leq H \leq 033$ can be formulated as follows:

- (1) Set $c(A) = b(A) \vee c(H)$ whereby bits A16 and A15 are XOR'd with bit H15. Retain c(H).

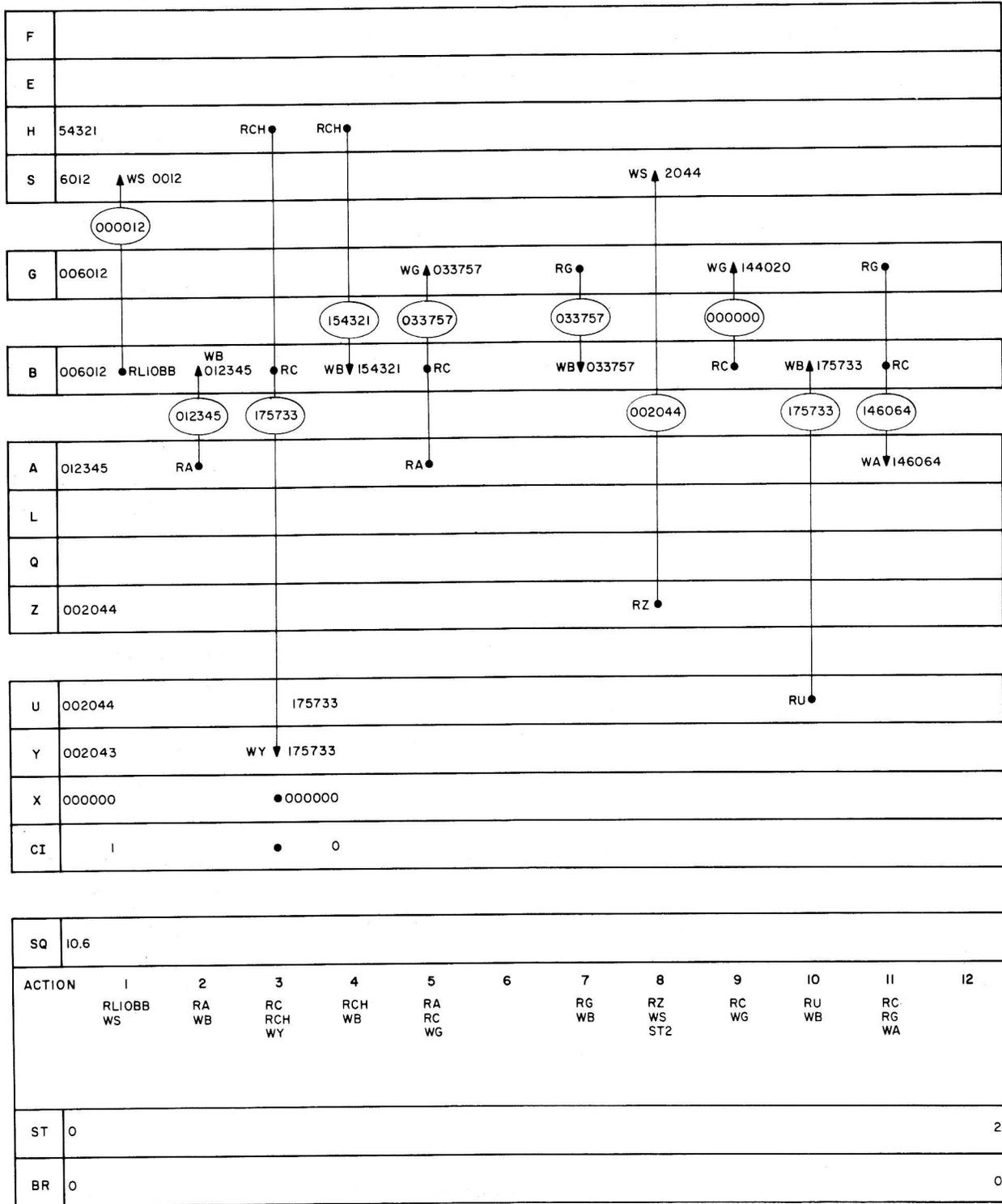


Figure 32-51. Subinstruction RXOR0

- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction RXOR H and j being the instruction stored at location $(I+1)$.
 Set $c(S) =$ relevant address of j .
 Set $c(SQ) =$ order code of j .
- (3) Set $c(Z) = b(Z) + 1 = I+2$.
- (4) Restore $c(I+1) = b(I+1)$ if $(I+1)$ represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-255. Special Cases of RXOR H:

- a. RXOR L and RXOR Q make use of all sixteen bits in L or Q to form the Exclusive OR.
- b. RXOR 003 and RXOR 004 can be used to form an Exclusive OR with the fourteen bits in SCALER2 or SCALER1.
- c. Instructions RXOR H with $005 \leq H \leq 033$ follow the rules of paragraph 32-254.
- d. RXOR 034 and RXOR 035 cannot form an Exclusive OR because downlink channels cannot be read by a Channel Instruction; $b(A)$ is retained in A.

32-256. Instruction RXOR H in reality performs $b(A) \vee \bar{c}(H) = m$, $\bar{b}(A) \vee c(H) = n$, $c(A) = \bar{m} \vee \bar{n}$, which supplies the same result. When instruction RXOR H is executed, action 1 of subinstruction RXOR0 (row 48 of table 32-4) replaces the quantity in register S by a 10 bit address, thus erasing two bits of the eighth code. Actions 2 and 3 form $\bar{b}(A) \vee c(H) = n$ which is entered into the Adder. Actions 4 and 5 form $b(A) \vee \bar{c}(H) = m$ which is entered into register G. Actions 7 and 9 form \bar{m} which is stored in register G, and actions 10 and 11 form $\bar{m} \vee \bar{n}$ which is entered into register A. Action 8 enters the address of the next instruction into register S and subinstruction STD2 calls forward the next instruction as usual.

32-257. Figure 32-51 illustrates the execution of subinstruction RXOR0 of instruction RXOR 012, XOR'ing the quantities 012345 and 154321 and providing 146064.

32-258. SPECIAL INSTRUCTIONS

32-259. INSTRUCTION EXTEND

32-260. Instruction EXTEND is a Special Instruction which is represented by order code 00.0006. Instruction EXTEND causes the execution of subinstruction STD2 which takes one MCT.

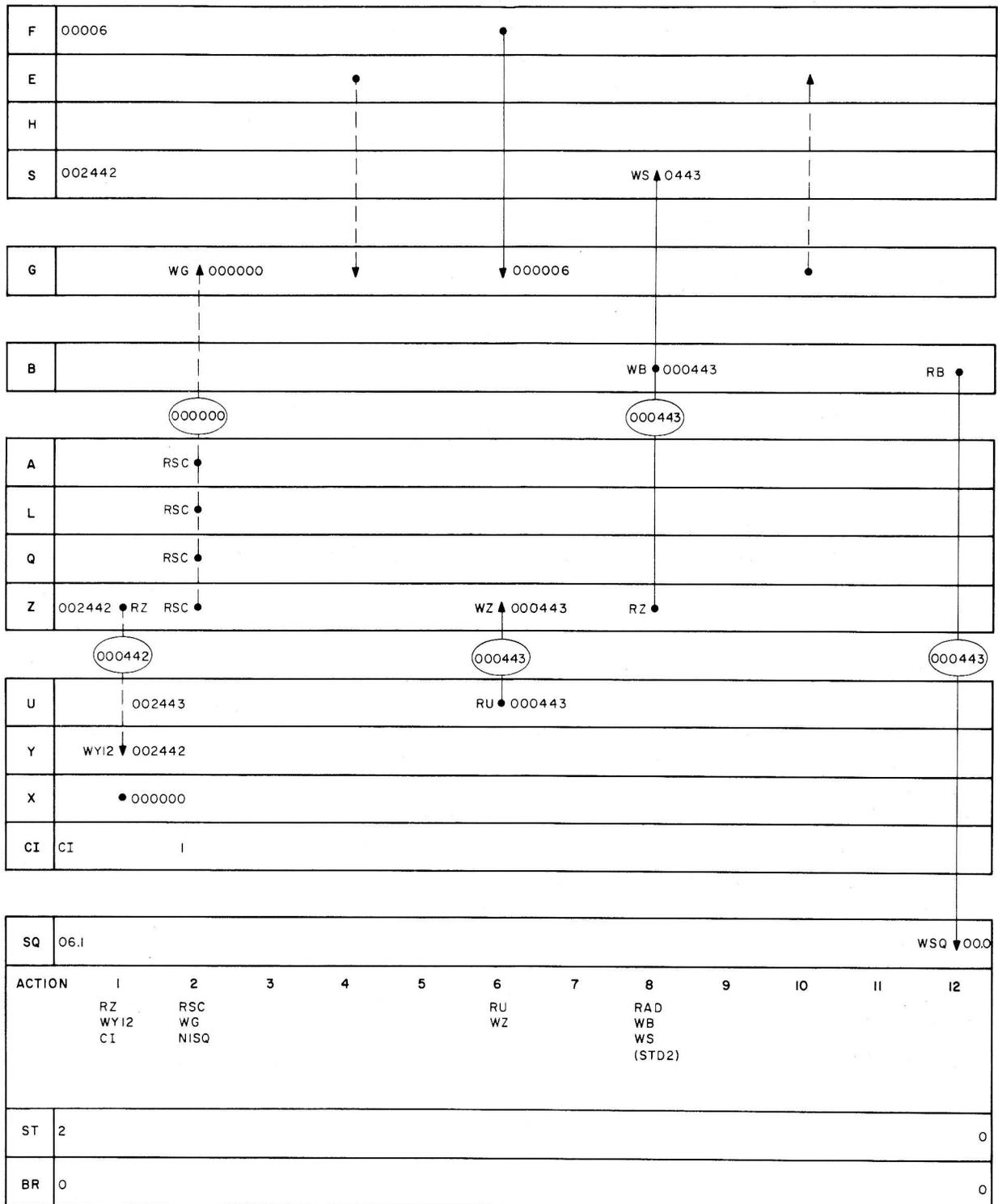


Figure 32-52. Subinstruction STD2, Preceding Instruction EXTEND

32-261. Instruction EXTEND enters a ONE into bit position EXT of register SQ to execute next an Extra Code Instruction. The operation EXTEND can be formulated as follows:

- (1) Enter a ONE into bit position EXT of register SQ and set flip-flop INHINT/RELINT.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction RELINT, and j being the instruction stored at location I+1.
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.
- (3) Set $c(Z) = c(Z+1) = I+2$.
- (4) Restore $c(I+1)$ if (I+1) represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-262. As the instruction preceding EXTEND is executed, its last subinstruction (STD2, TC0, BZF0, BZMF0, MP3, or RSM3) enters the order code of instruction EXTEND into register G at time 2, 4, or 6. (Compare figure 32-52 with figures 32-1, 32-2, 32-8, and 32-27.) Register G recognizes the presence of code 0.0006 and control pulses RZ and ST2 are generated at time 8 instead of control pulse RG in reply to control pulse RAD. Furthermore, a ONE is entered into bit position EXT of register SQ, and flip-flop INHINT/RELINT is set. Action 8 transfers the address of the next instruction from register Z to registers B and S, and the order code 00.0 is transferred to register SQ at time 12. Thereafter, subinstruction STD2 is executed to call forward the Extra Code Instruction following instruction EXTEND. During the execution of the last subinstruction of the Extra Code Instruction, except for instruction NDX K, bit position EXT and flip-flop INHINT/RELINT are reset.

32-263. INSTRUCTION INHINT

32-264. Instruction INHINT (Inhibit Interrupt) is a Special Instruction which is represented by order code 00.0004. Instruction INHINT causes the execution of subinstruction STD2 which takes one MCT.

32-265. Instruction INHINT commands the Sequence Generator (SQG) to refuse to accept any request for the execution of instruction RUPT. The operation INHINT can be formulated as follows:

- (1) Set flip-flop INHINT/RELINT.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction RELINT, and j being the instruction stored at location I+1.
Set $c(S) =$ relevant address of j.
Set $c(SQ) =$ order code of j.

- (3) Set $c(Z) = c(Z+1) = I+2$.
- (4) Restore $c(I+1)$ if $(I+1)$ represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-266. The execution of instruction INHINT is similar to that of instruction EXTEND described in paragraph 32-262. When register G recognizes the presence of code 00.0004, control pulses RZ and ST2 are generated instead of control pulse RG at time 8, flip-flop INHINT/RELINT is set, but bit position EXT of register SQ is not set.

32-267. INSTRUCTION RELINT

32-268. Instruction RELINT (Release Interrupt Inhibit) is a Special Instruction which is represented by order code 00.0003. Instruction RELINT causes the execution of subinstruction STD2 which takes one MCT.

32-269. Instruction RELINT commands the Sequence Generator (SQG) to accept any request for the execution of a RUPT instruction. The operation RELINT can be formulated as follows:

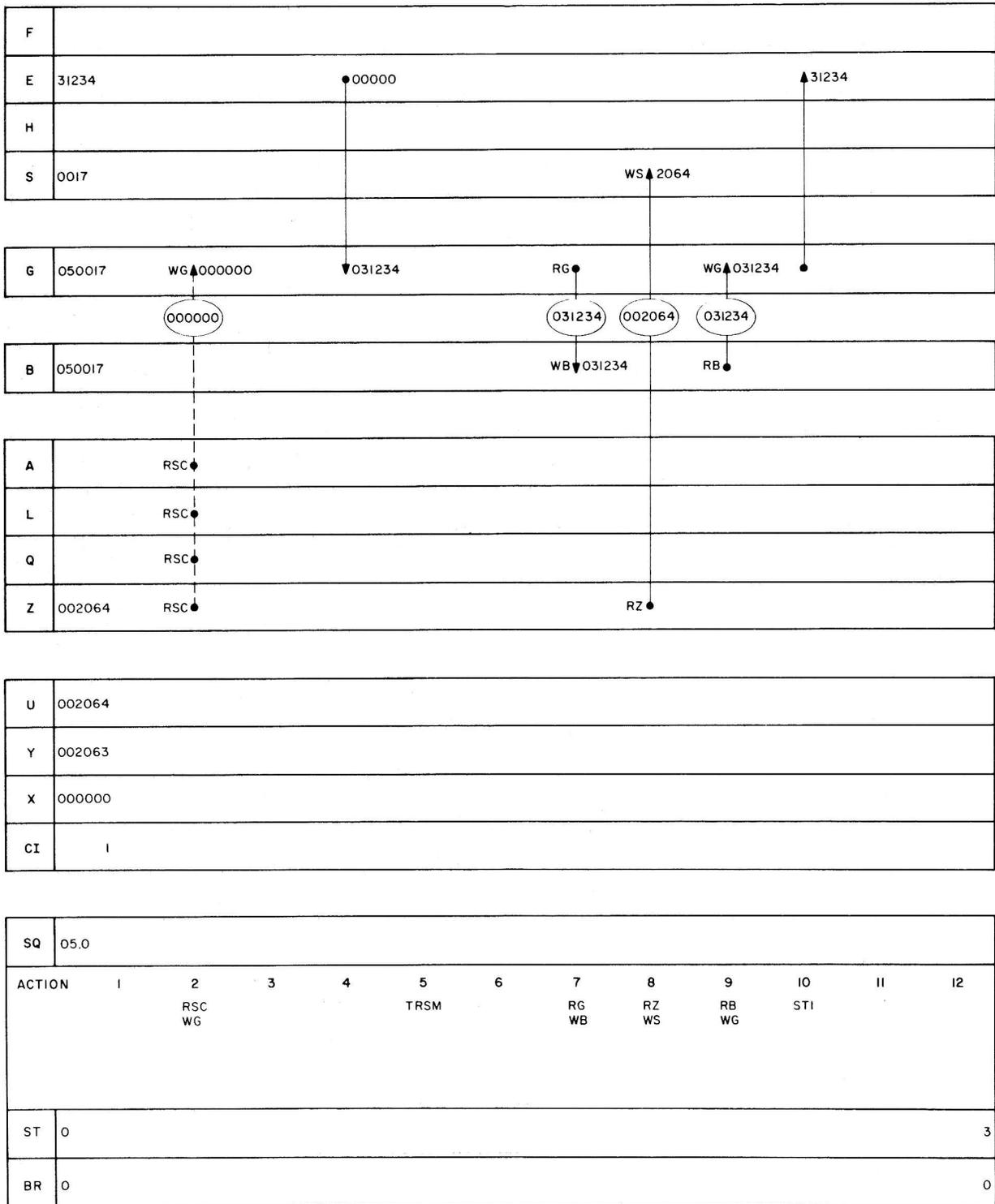
- (1) Reset flip-flop INHINT/RELINT.
- (2) Set $c(B) = c(I+1) = j$, I being the address of instruction RELINT, and j being the instruction stored at location $I+1$.
Set $c(S) =$ relevant address of j .
Set $c(SQ) =$ order code of j .
- (3) Set $c(Z) = c(Z+1) = I+2$.
- (4) Restore $c(I+1)$ if $(I+1)$ represents an address in E Memory.

Point (2) implies that instruction j is executed next.

32-270. The execution of instruction RELINT is similar to that of instructions EXTEND and INHINT described in paragraphs 32-262 and 32-266. When register G recognizes the presence of code 00.0003, control pulses RZ and ST2 are generated instead of control pulse RG at time 8, flip-flop INHINT/RELINT is reset, and bit position EXT of register SQ is not set.

32-271. INSTRUCTION RESUME

32-272. Instruction RESUME (Resume Interrupted Program) is a Special Instruction which is represented by order code 05.0017. Instruction RESUME consists of subinstruction NDX0 and RSM3, the execution of which takes two MCT's.



2716A

Figure 32-53. Subinstruction NDX0 of Instruction RESUME

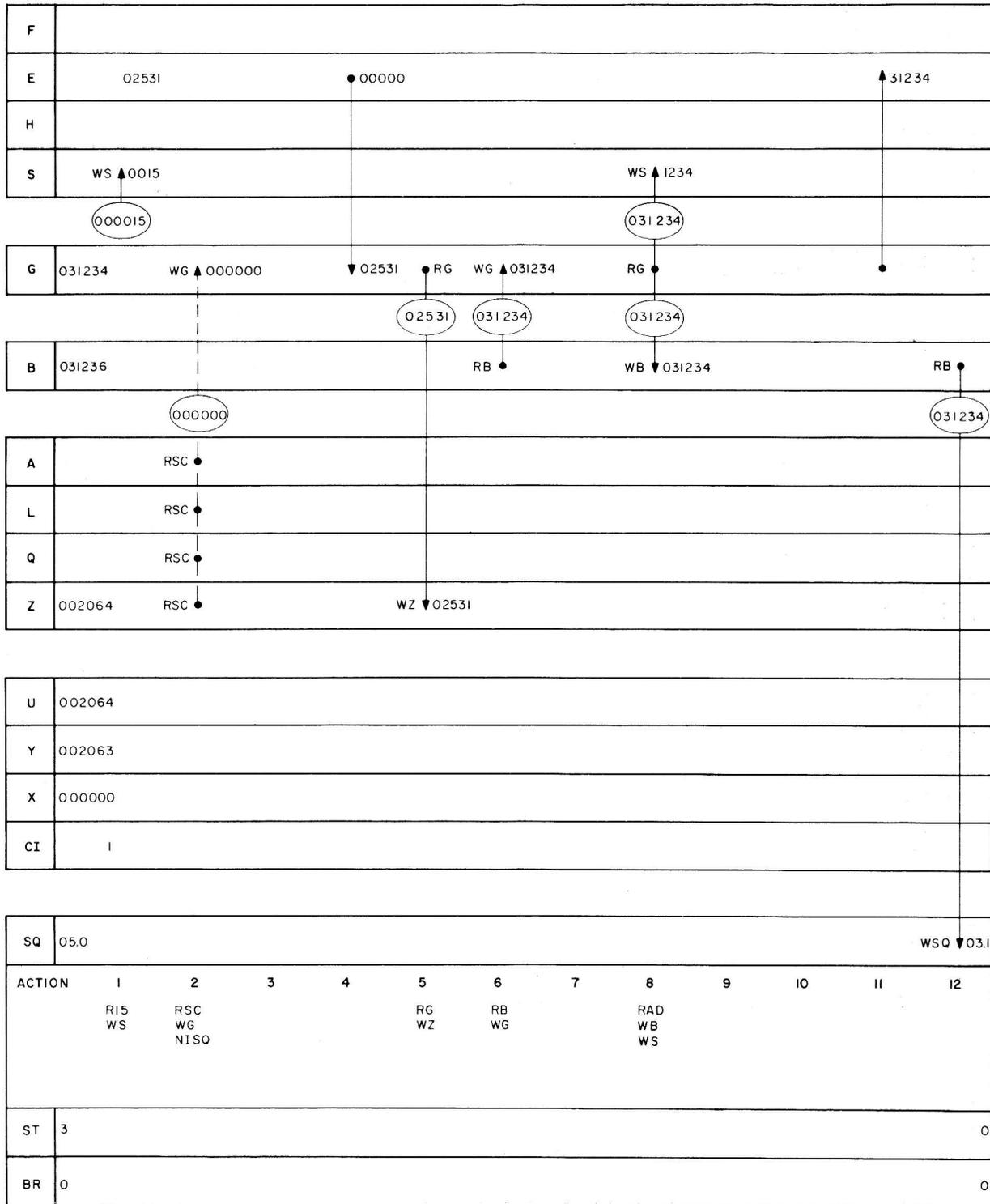


Figure 32-54. Subinstruction RSM3

32-273. Instruction RESUME commands the Sequence Generator to accept any RUPT request, returns pertinent data of the interrupted program from memory to CP registers, and resumes the execution of the interrupted program section if no RUPT request has been made. The operation RESUME can be formulated as follows:

- (1) Reset flip-flop INHINT/RELINT.
- (2) Set $c(B) = c(BRUPT) = c(0017)$.
Set $c(S) =$ relevant address of instruction contained in B.
Set $c(SQ) =$ order code of instruction contained in B.
- (3) Set $c(Z) = b(ZRUPT) = c(0015)$.
- (4) Restore $c(BRUPT) = b(BRUPT)$.
Set $c(ZRUPT) = c(BRUPT)$.

Point (2) implies that the instruction which was stored in BRUPT will be executed next.

32-274. At the time instruction RUPT was executed the last time, flip-flop IIP was set to prevent the interruption of an interrupting program section, the next instruction of the interrupted program section was transferred from register B to location BRUPT = 0017, and the address of the second-next instruction was transferred from register Z to location ZRUPT = 0015. (Refer to paragraphs 30-123 and 32-282.) By returning $c(BRUPT)$ and $c(ZRUPT)$ to registers B and Z, respectively, the execution of the interrupted program section is continued. Subinstruction NDX0 of instruction RESUME (NDX 0017) returns the $c(BRUPT)$ to register B. Subinstruction RSM3 returns $c(ZRUPT)$ to register Z and enters the relevant address and the order code of the returned instruction now contained in register B into registers S and SQ, respectively. (Refer to row 49 of table 32-4 and figures 32-53 and 32-54.)

32-275. INSTRUCTIONS CYR, SR, CYL and EDOP

32-276. Instruction CYR (Cycle Right) is a Special Instruction which is represented by code .0020, SR (Shift Right) by .0021, CYL (Cycle Left) by .0022, and EDOP (Edit Operator) by .0023. These codes can be used with most Basic and Extra Code Instructions. Whenever one of these codes is used, the quantity being entered into register G is edited as listed below (refer to paragraph 30-41 and table 30-1).

CYR cycled one place to the right
SR shifted one place to the right
CYL cycled one place to the left
EDOP shifted seven places to the right

The effect of codes .0020 through .0023 is described under special cases with each Basic or Extra Code Instruction.



32-277. INVOLUNTARY INSTRUCTIONS

32-278. INTERRUPTING INSTRUCTIONS

32-279. INSTRUCTION RUPT

32-280. Instruction RUPT (Interrupt Program Execution) is an Interrupting Instruction which is executed at the occurrence of certain events (paragraph 30-131) by entering order code 10.7 into register SQ. Instruction RUPT initiates certain programmed operations (paragraph 30-132 and table 30-6) and consists of subinstructions RUPT0, RUPT1, and STD2, the execution of which takes three MCT's. The execution of instruction RUPT is inhibited if a RUPT was executed after the last RESUME or if an INHINT was executed after the last RELINT.

32-281. Instruction RUPT commands the Sequence Generator (SQG) to refuse to accept any other RUPT request (until instruction RESUME is executed), transfers pertinent data of the program section being interrupted to memory, and transfers program control to the requested programmed operation. The operation RUPT can be formulated as follows:

- (1) Set flip-flop INHINT/RELINT.
- (2) Set $c(\text{BRUPT}) = c(0017) = c(\text{B})$.
- (3) Set $c(\text{ZRUPT}) = c(0015) = c(\text{Z})$.
- (4) Set $c(\text{Z}) =$ address of RUPT Transfer Routine provided by Interrupt Priority Control.
- (5) Set $c(\text{B}) =$ first instruction of desired RUPT Transfer Routine.
Set $c(\text{S}) =$ relevant address of instruction contained in B.
Set $c(\text{SQ}) =$ order code of instruction contained in B.

Point (5) implies that the first instruction of the desired RUPT Transfer Routine will be executed next.

32-282. When instruction RUPT is executed, subinstruction RUPT0 (row 50 of table 32-4 and figure 32-55) transfers $c(\text{Z})$ to location $\text{ZRUPT} = 0015$ in memory (table 30-4). Subinstruction RUPT1 (row 51 of table 32-4 and figure 32-56) transfers $c(\text{B})$ to location $\text{BRUPT} = 0017$, and enters the address of the RUPT Transfer Routine into registers Z and S. Subinstruction STD2 calls forward the first instruction of the RUPT Transfer Routine.

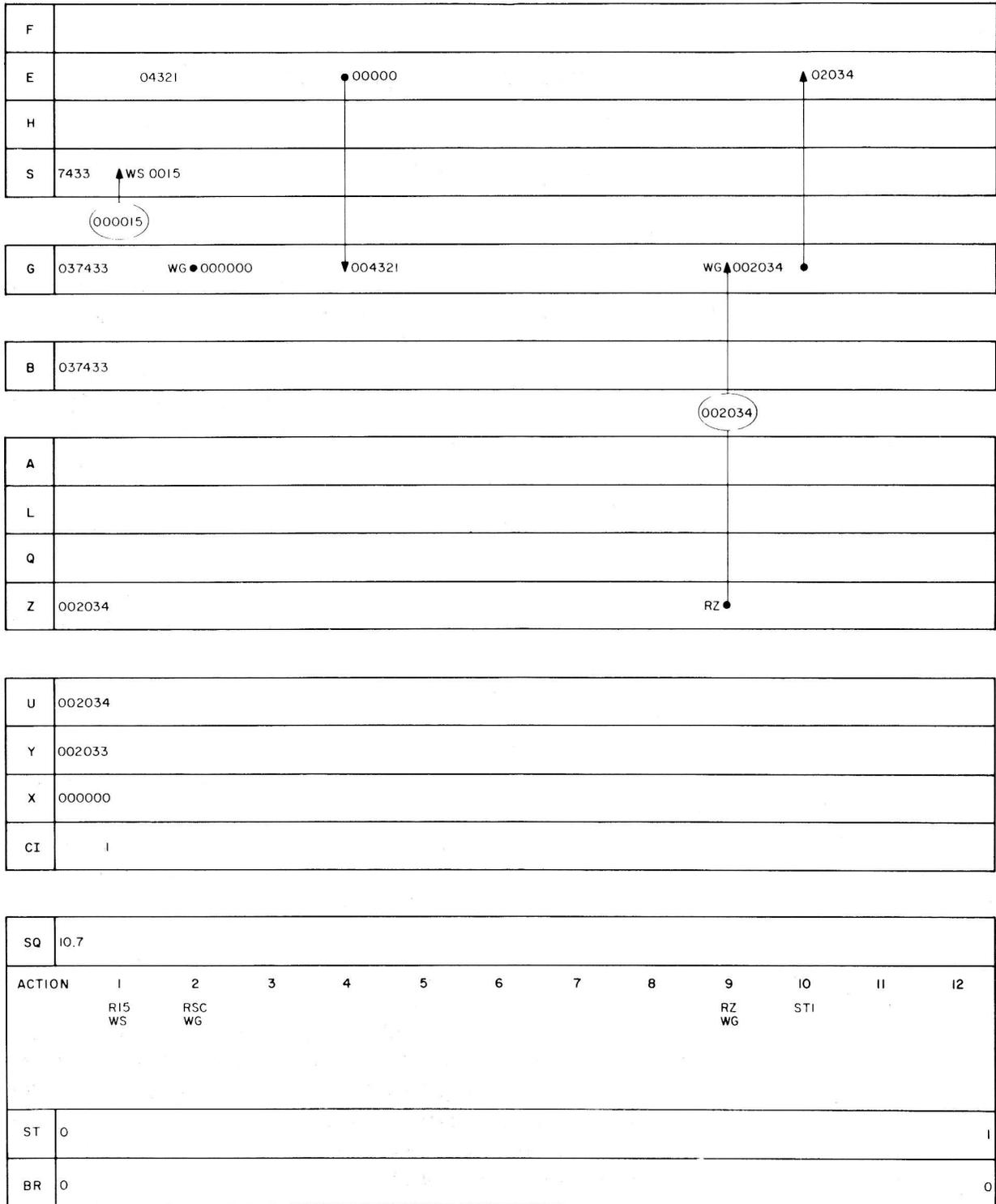
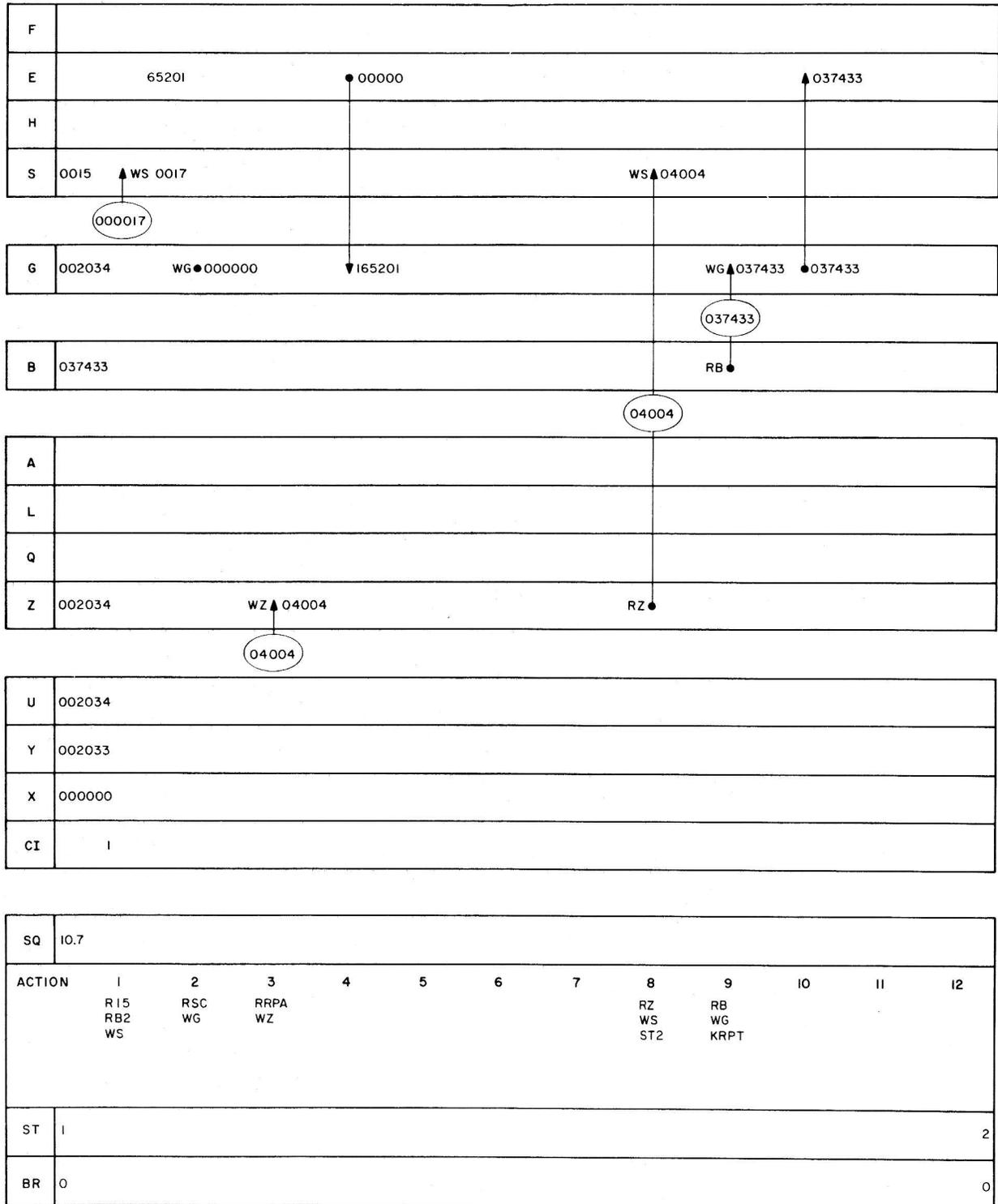


Figure 32-55. Subinstruction RUPT0



2851A

Figure 32-56. Subinstruction RUPT1

32-283. INSTRUCTION GO

32-284. Instruction GO is an Interrupting Instruction which is executed at the occurrence of certain errors (if signal GOJAM is generated) by entering order code 00. into register SQ and entering 1 into the stage counter (ST). Instruction GO initiates the execution of the restart sequence (table 30-6) and consists of subinstructions GOJ1 and TC0, the execution of which takes two MCT's.

32-285. Instruction GO enters TC 4000 into registers B and S, 4000 being the address of RUPT Transfer Routine GO. (Refer to row 52 of table 32-4.) Instruction TC 4000 is executed after instruction GO.

32-286. COUNTER INSTRUCTIONS

32-287. INSTRUCTION PINC C

32-288. Instruction PINC C (Plus Increment C) is a Counter Instruction which is executed at the occurrence of certain events (paragraph 30-137) without entering an order code into register SQ and is independent of the content of register SQ. Instruction PINC C consists of subinstruction PINC, the execution of which takes one MCT.

32-289. Instruction PINC C increments by one the content of that E Memory counter C the address of which is supplied by the Counter Priority Control. The operation PINC C can be formulated as follows:

- (1) Set $c(C) = b(C) + 1$ except for overflow bit.
- (2) Retain $c(B)$.
Retain $c(S)$.
Retain $c(SQ)$.
- (3) Retain $c(Z)$.

Point (2) implies that the instruction stored in B is executed next.

32-290. Special Cases of PINC C.

- a. PINC 0024 causes the execution of PINC 0025 in case of overflow of $c(0024)$.
- b. PINC 0026, PINC 0027, or PINC 0030 causes execution of RUPT in case of an overflow.

32-291. When instruction PINC C is executed, action 1 of subinstruction PINC (row 53 of table 32-4) enters into register S the counter address C provided by the Counter Priority Control.

The content of the addressed counter is transferred to register G at time 4. Actions 5, 6, and 7 add the quantity 000001 to the content of register G. At time 10, the incremented quantity is returned to the addressed counter in E Memory. Action 8 re-enters into register S the relevant address contained in register B to establish the original conditions.

32-292. Figure 32-57 illustrates the execution of instruction PINC 0025. Counter 0025 contains quantity 00102 which becomes 00103 after being incremented. If the counter contained 37777, it would contain 00000 after being incremented, and the execution of PINC 0024 would be requested. If the addressed counter was counter 0026, 0027, or 0030, the execution of instruction RUPT would be requested in case of overflow of the addressed counter.

32-293. INSTRUCTION MINC C

32-294. Instruction MINC C (Minus Increment C) is a Counter Instruction which is executed at the occurrence of certain events (paragraph 30-137) without entering an order code into register SQ and is independent of the content of register SQ. Instruction MINC C consists of subinstruction MINC, the execution of which takes one MCT.

32-295. Instruction MINC C decrements by one the content of that E Memory counter C the address of which is supplied by the Counter Priority Control. The operation MINC C can be formulated as follows:

- (1) Set $c(C) = b(C) + 1$ except for overflow bit.
- (2) Retain $c(B)$.
Retain $c(S)$.
Retain $c(SQ)$.
- (3) Retain $c(Z)$.

Point (2) implies that the instruction stored in B is executed next.

32-296. There are no special cases of MINC C. Instructions PINC C and MINC C are identical except for action 6 (compare rows 53 and 54 of table 32-4). Control pulse MONEX of subinstruction MINC replaces PONEX of subinstruction PINC.

32-297. INSTRUCTION DINC C

32-298. Instruction DINC C (Diminish Increment C) is a Counter Instruction which is executed at the occurrence of certain events (paragraph 30-137) without entering an order code into register SQ and is independent of the content of register SQ. Instruction DINC C consists of subinstruction DINC, the execution of which takes one MCT.

32-299. Instruction DINC C diminishes (decreases magnitude) by one the content of that E memory drive counter C the address of which is supplied by the Counter Priority Control. The operation DINC C can be formulated as follows:

- (1) If $c(C)$ is positive non-zero, set $c(C) = b(C) - 1$, and generate one plus drive pulse.
If $c(C)$ is negative non-zero, set $c(C) = b(C) + 1$ and generate one minus drive pulse.
If $c(C)$ is plus or minus zero, set $c(C) = b(C)$ and generate no drive pulse.
- (2) Retain $c(B)$.
Retain $c(S)$.
Retain $c(SQ)$.
- (3) Retain $c(Z)$.

Point (2) implies that the instruction stored in B is executed next.

32-300. There are no special cases of DINC. Instruction DINC C is identical to instructions PINC C and MINC C except for action 6. (Compare rows 53 through 55 of table 32-4.) The main difference is that instruction DINC C is used with drive operations as described in paragraphs 30-90 through 30-104.

32-301. INSTRUCTION PCDU C

32-302. Instruction PCDU C (Plus CDU C) is a Counter Instruction which is executed at the occurrence of certain events (paragraph 30-137) without entering an order code into register SQ and is independent of the content of register SQ. Instruction PCDU C consists of subinstruction PCDU, the execution of which takes one MCT.

32-303. Instruction PCDU C increments by one the content of that CDU counter C in E Memory the address of which is supplied by the Counter Priority Control. The incrementing is carried out in TWO's complement arithmetic since CDU counters contain cyclic TWO's complement numbers (paragraphs 30-46 and 30-47). The operation PCDU C can be formulated as follows:

- (1) Set $c'(C) = b'(C) + 1$ where $c'(C)$ and $b'(C)$ are cyclic TWO's complement numbers.
- (2) Retain $c(B)$.
Retain $c(S)$.
Retain $c(SQ)$.
- (3) Retain $c(Z)$.

Point (2) implies that the instruction stored in B is executed next.

32-304. There are no special cases of PCDU C. Instruction PCDU C is similar to instruction PINC C. (Compare rows 53 and 56 of table 32-4.) Control pulse PONEX of subinstruction PINC is replaced by CI of PCDU, and RU is replaced by RUS. If the ONE's complement quantity 37777 (plus 37777) contained in a counter is incremented by PINC C, the resulting quantity is 00000 (plus zero) because the overflow bit was lost during the storing of the incremented quantity. If the TWO's complement quantity 37777 (nearly 180° as shown in paragraph 32-209) is incremented by PCDU C, the resulting quantity is 40000 (180°) because of control pulse RUS which placed the overflow bit into bit position 15 of the counter. If the ONE's complement quantity 77777 (minus zero) is incremented by PINC C, the resulting quantity is 00001 (plus one). If the cyclic TWO's complement quantity 77777 (maximum, i. e. nearly 360°) is incremented by PCDU, the resulting quantity is 00000 (360° or zero).

32-305. INSTRUCTION MCDU C

32-306. Instruction MCDU C (Minus CDU C) is a Counter Instruction which is executed at the occurrence of certain events (paragraph 30-137) without entering an order code into register SQ and is independent of the content of register SQ. Instruction MCDU C consists of subinstruction MCDU, the execution of which takes one MCT.

32-307. Instruction MCDU C decrements by one the content of that CDU counter C in E Memory the address of which is supplied by the Counter Priority Control. The decrementing is carried out in TWO's complement arithmetic since CDU counters contain cyclic TWO's complement numbers (paragraphs 30-46 and 30-47). The operation MCDU C can be formulated as follows:

- (1) Set $c'(C) = b'(C) - 1$ where $c'(C)$ and $b'(C)$ are cyclic TWO's complement numbers.
- (2) Retain $c(B)$.
Retain $c(S)$.
Retain $c(SQ)$.
- (3) Retain $c(Z)$.

Point (2) implies that the instruction stored in B is executed next.

32-308. There are no special cases of MCDU C. Instruction MCDU C is similar to instructions MINC C and PCDU C. (Compare rows 53, 56, and 57 of table 32-4.) Action 6 of subinstruction MCDU consists of control pulses MONEX and CI which together add the TWO's complement quantity 77777 (minus one) to 00000 (zero or 360°) if this quantity is contained in an addressed counter. If a counter contains 40000 (180°) 37777 is contained after the decrementing due to control pulse RUS.

32-309. INSTRUCTION SHINC C

32-310. Instruction SHINC (Shift Increment C) is a Counter Instruction which is executed at the occurrence of certain events (paragraph 30-137) without entering an order code into register SQ and is independent of the content of register SQ. Instruction SHINC C consists of subinstruction SHINC, the execution of which takes one MCT.

32-311. Instruction SHINC C shifts one place to the left the content of that E Memory counter (0045 or 0046) the address of which is supplied by the Counter Priority Control. The operation SHINC C can be formulated as follows:

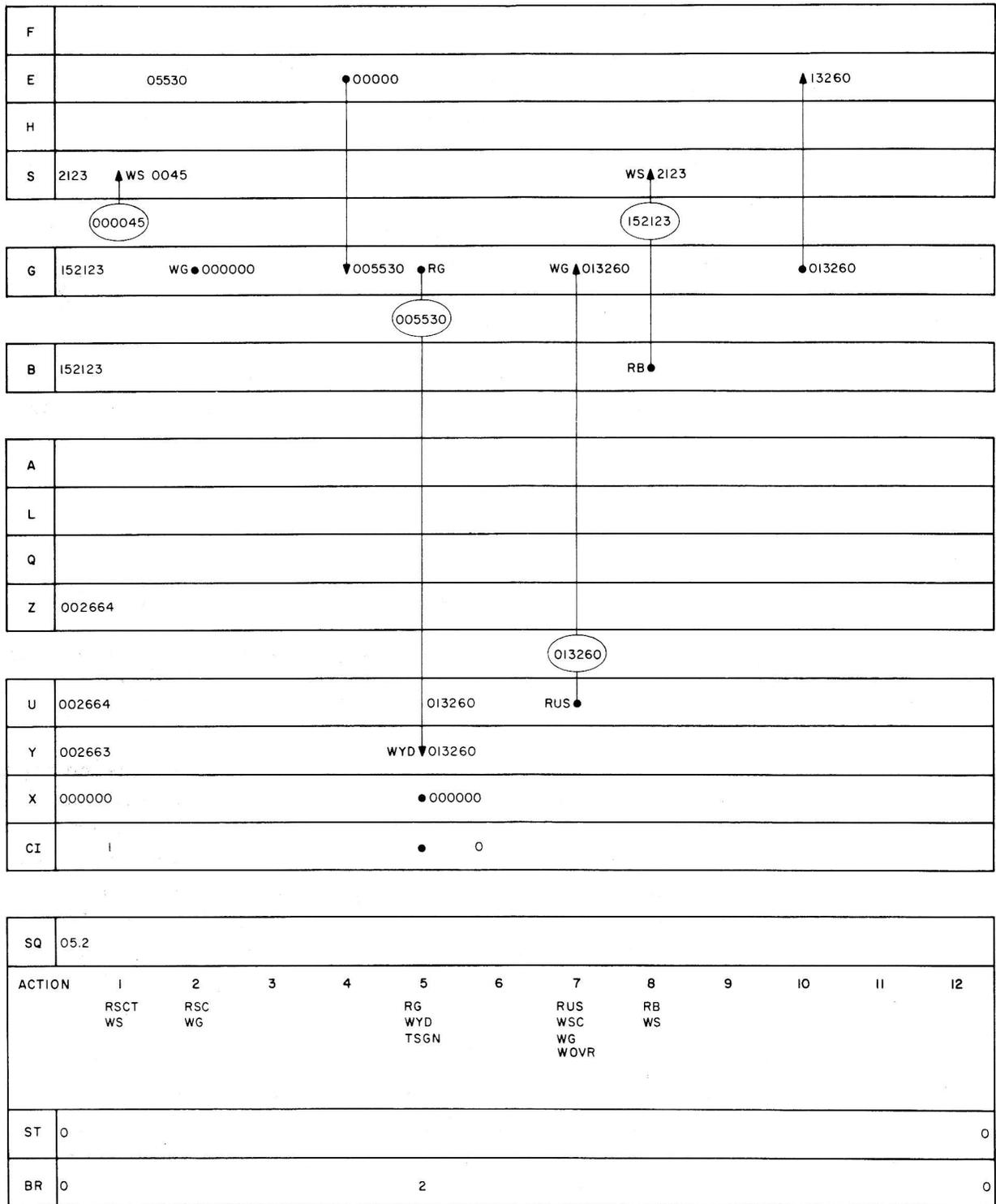
- (1) Set $c(C) = 2b(C)$ where $b(C)$ is always a positive quantity and $c(C)$ includes an overflow bit (instead of a sign bit) in bit position 15 in case of overflow.
- (2) Retain $c(B)$.
Retain $c(S)$.
Retain $c(SQ)$.
- (3) Retain $c(Z)$.

Point (2) implies that the instruction stored in B is executed next.

32-312. Instruction SHINC C is used for serial to parallel conversion. If SHINC 0045 is executed and an overflow occurs, the execution of instruction RUPT is requested.

32-313. When instruction SHINC C is executed, action 1 of subinstruction SHINC (row 58 of table 32-4) enters into register S the counter address C provided by the Counter Priority Control. The content of the addressed counter is transferred to register G at time 4. Action 5 doubles the quantity and enters this doubled quantity into the Adder. Action 7 enters the doubled quantity into register G whereby any overflow bit is entered into bit positions 16 and 15 of G. At time 10 the content of register G is entered into the addressed counter. Action 8 re-enters into register S the relevant address contained in register B to re-establish the original conditions.

32-314. Figure 32-58 illustrates the execution of instruction SHINC 0045. Originally, counter 0045 which contained 05530 before the shifting operation, contains 013260 after the shifting operation. If quantity 25530 were contained originally, 53230 would be contained after shifting and the execution of instruction RUPT would be requested.



2857A

Figure 32-58. Subinstruction SHINC

32-315. INSTRUCTION SHANC C

32-316. Instruction SHANC C (Shift and Add Increment C) is a Counter Instruction which is executed at the occurrence of certain events (paragraph 30-137) without entering an order code into register SQ and is independent of the content of register SQ. Instruction SHANC C consists of subinstruction SHANC C, the execution of which takes one MCT.

32-317. Instruction SHINC C shifts one place to the left the content of that E Memory counter (0045 or 0046) the address of which is supplied by the Counter Priority Control and adds a ONE into bit position 1. The operation SHANC C can be formulated as follows:

- (1) Set $c(C) = 2b(C)+1$ where $b(C)$ is always a positive quantity and $c(C)$ includes an overflow bit (instead of a sign bit) in bit position 15 in case of overflow.
- (2) Retain $c(B)$.
Retain $c(S)$.
Retain $c(SQ)$.
- (3) Retain $c(Z)$.

Point (2) implies that instruction stored in B is executed next.

32-318. Instruction SHANC C is also used for serial to parallel conversion similarly to instruction SHINC C. (Compare rows 58 and 59 of table 32-4.) Control pulse CI of action 5 adds the ONE into bit position 1 of the Adder; this ONE is later transferred to bit position 1 of the counter. If SHANC 0045 is executed and an overflow occurs, the execution of instruction RUPT is requested.

32-319. PERIPHERAL INSTRUCTIONS

32-320. SEQUENCE CHANGING TEST INSTRUCTIONS

32-321. INSTRUCTION TCSAJ K

32-322. Instruction TCSAJ K (Transfer Control to Specified Address K) is a test instruction which is executed on command from Ground Support Equipment (GSE) such as the Computer Test Set (CTS) or the Program Analyzer Console (PAC). The address K is supplied by the CTS or PAC. Instruction TCSAJ K consists of subinstructions TCSAJ3 and STD2, the execution of which takes two MCT's.

32-323. Instruction TCSAJ K takes the next instruction from location K. The operation TCSAJ K with $0024 \leq K \leq 7777$ can be formulated as follows:

- (1) Retain $c(Q)$.
- (2) Set $c(B) = c(K) = k$, k being the instruction stored at location K.
Set $c(S) =$ relevant address of k .
Set $c(SQ) =$ order code of k .
- (3) Set $c(Z) = K+1$.
- (4) Restore $c(K) = b(K)$ if K represents an address in E Memory

Point (2) implies that instruction k is executed next.

32-324. The special cases of TCSAJ K are the same as for TC K (paragraph 32-35). Instruction TCSAJ K is similar to instructions TC K and TCF F in effect but differs in the number of subinstructions. Action 8 of subinstruction TCSAJ3 (row 60 of table 32-4) enters the address K supplied by the GSE into registers S and Z. Subinstruction STD2 then increments by one the content of register Z and calls forward the instruction located at K.

32-325. DISPLAY AND LOAD TEST INSTRUCTIONS

32-326. INSTRUCTION FETCH K

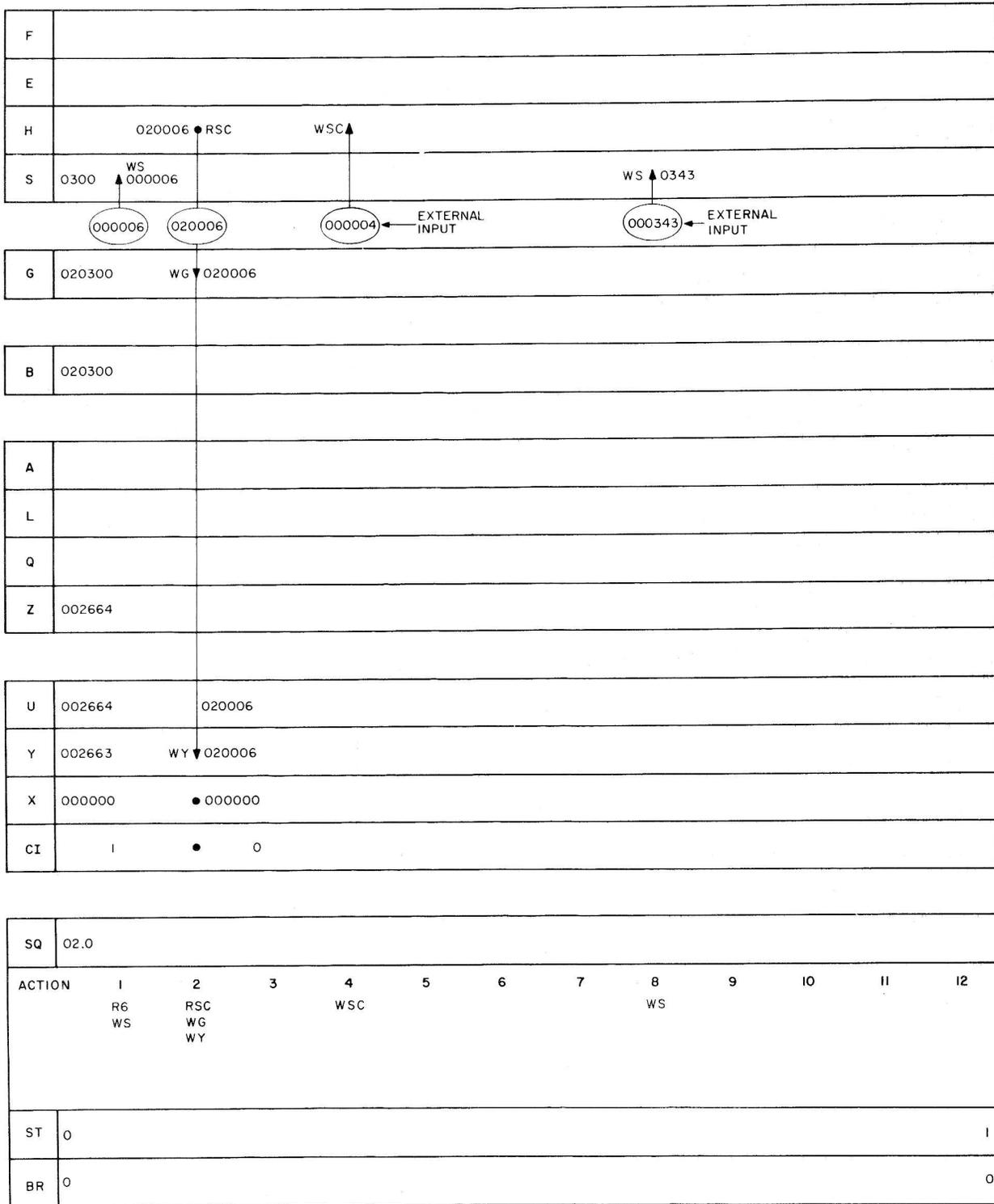
32-327. Instruction FETCH K is a display instruction which is executed on command of the GSE. The address K is supplied by the GSE. Instruction FETCH K consists of subinstructions FETCH0 and FETCH1, the execution of which takes two MCT's.

32-328. Instruction FETCH K enters into either register EBANK or FBANK, a bank number received from the GSE; enters into register S an address K received from the GSE, and provides for display at the WA's the content of that location K and the final content of BBANK. Thereafter, the before contents of BBANK and S are restored. The operation of instruction FETCH K is illustrated on figures 32-59 and 32-60. Action 1 of subinstruction FETCH0 (row 61 of table 32-4) enters the quantity 000006 into register S to simultaneously address registers EBANK and FBANK. In the example, register FBANK contains F bank number 1 (a ONE in bit position 11 as shown in table 30-3) and register EBANK contains E bank number 6 (ONE's in bit positions 11 and 10 as shown in table 30-2). If both registers are addressed and their contents are read into the WA's, the quantity 020006 appears because the content of register EBANK is shifted eight places to the right (table 30-1). Action 2 stores this quantity in the Adder. Action 4 enters a new bank number into register EBANK or FBANK or both. Action 8 enters the address of the required location (0343) into register S. At time 4 of subinstruction FETCH1 (row 62 of table 32-4) the content 76543 of location 0343 is entered into register G at time 4. (In case a CP register or an F Memory location is addressed, the quantity is entered into register G at time 2 or 6, respectively.) Action 7 places the same quantity into the WA's for display. Action 8 restores the original contents of register S, EBANK, and FBANK. Thus, program execution can be continued after the execution of instruction FETCH K. Restoring the contents of registers EBANK and FBANK may be inhibited. Action 10 places the content of registers EBANK and FBANK into the WA's for display. At time 10, the original content of location K is restored.

32-329. INSTRUCTION STORE E

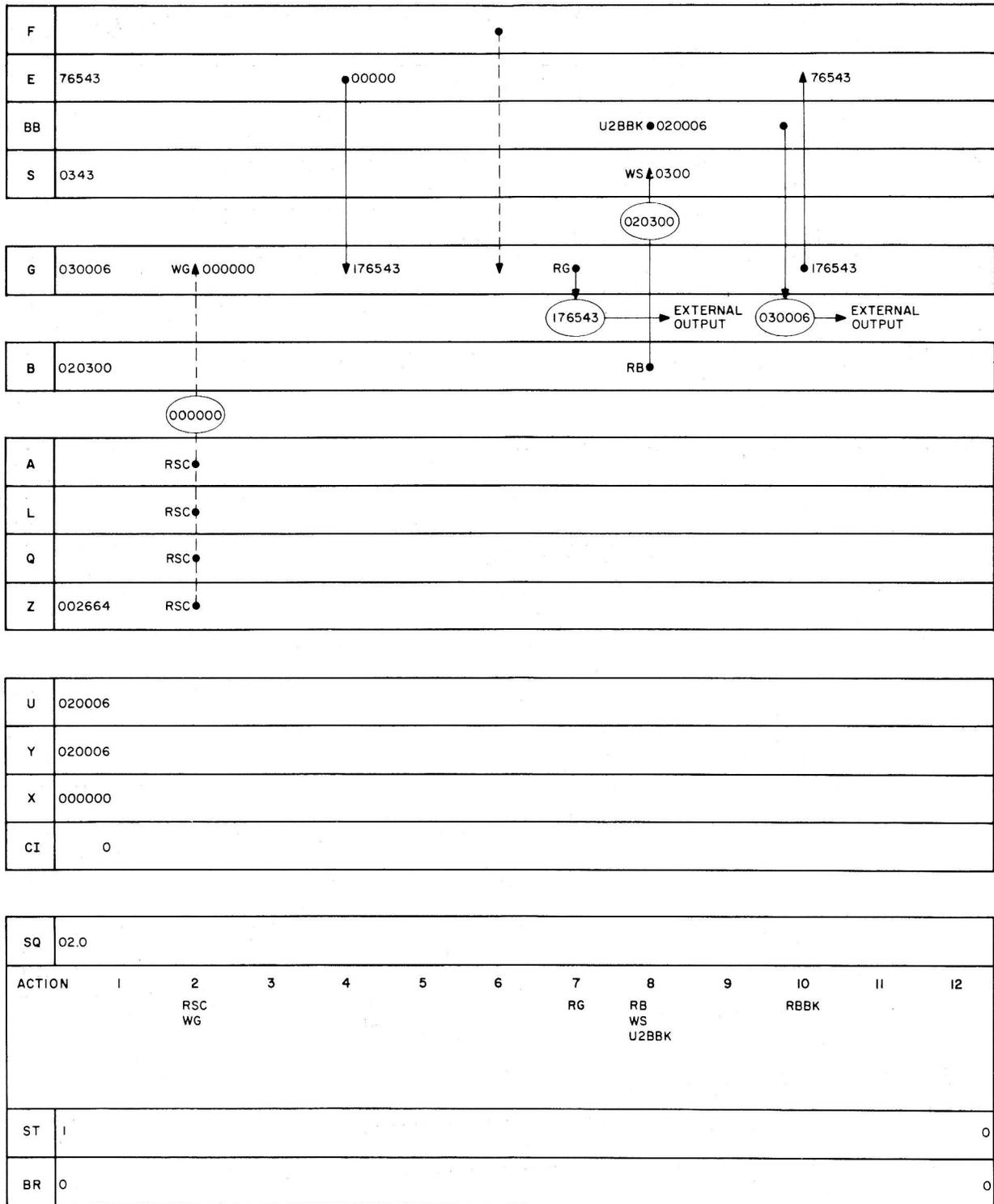
32-330. Instruction STORE E is a load instruction which is executed on command of the GSE. The address E is supplied by the GSE. Instruction STORE E consists of subinstructions STORE0 and STORE1, the execution of which takes two MCT's.

32-331. Instruction STORE E enters into either register EBANK or FBANK, a bank number received from the GSE, enters into register S an address E received from the GSE, and enters into that location E the quantity provided by the GSE. Thereafter the before content of BBANK is restored unless $E = BBANK$. The before content of S is always restored. The operation of instruction STORE E is similar to that of instruction FETCH K. (Compare rows 62 and 63 of table 32-4 with rows 61 and 62.) Subinstruction STORE0 is identical to subinstruction FETCH0 in that both address a specific location. Action 4 and action 9 of subinstruction STORE1 enter the quantity, which is entered into the WA's by the GSE into the addressed location. If $0020 \leq E \leq 0023$, the quantity entered is edited.



2864A

Figure 32-59. Subinstruction FETCH0



2865A

Figure 32-60. Subinstruction **FETCH1**

32-332. INSTRUCTION INOTRD H

32-333. Instruction INOTRD H is a channel display instruction which is executed on command of the GSE. The channel address H is supplied by the GSE. Instruction INOTRD H consists of subinstruction INOTRD, the execution of which takes one MCT.

32-334. Instruction INOTRD H provides at the WA's for display the content of channel H, which is specified by the GSE. Action 1 of subinstruction INOTRD (row 65 of table 32-4) enters into register S the address of channel H, address H being supplied by the GSE. Action 5 enters the content of the addressed channel into the WA's for display. Action 8 restores the original content of register S.

32-335. INSTRUCTION INOTLD H

32-336. Instruction INOTRD H is a channel load instruction which is executed on command of the GSE. The channel address H is supplied by the GSE. Instruction INOTRD H consists of subinstruction INOTRD, the execution of which takes one MCT.

32-337. Instruction INOTLD enters into channel H, as specified by the GSE, the quantity provided by the GSE. The operation of instruction INOTLD is similar to that of instruction INOTRD. (Compare rows 65 and 66 of table 32-4.) Action 7 of subinstruction INOTLD enters the quantity provided into channel H.

