G-20 Central Processor Service Manual

VOLUME 2

TABLE OF CONTENTS

CENTRAL PROCESSOR SERVICE MANUAL

VOLUME II

	PAGE
CHAPTER 9 – MASTER CONTROL	9–1
SECTION 9.1 – INTRODUCTION	9–1
SECTION 9.2 – PROCESSING OF A SIMPLE OPCODE	9–8
SECTION 9.3 – TASKS PERFORMED BY KC	9–11
9.3–1 Command Access	9–11
9.3–2 Operand Assembly	9–13
9.3–3 Opcode Startup	9–14
SECTION 9.4 – BREAKDOWN WITHIN THE LOGIC	9–17
SECTION 9.5 – ENTRY AT KCA	9–22
SECTION 9.6 – ENTRY AT KZC	9–26
SECTION 9.7 – ENTRY AT KZE	9–33
SECTION 9.8 – ENTRY AT KCL	9–40
SECTION 9.9 – ENTRY AT KCS	9–47
SECTION 9.10 – REVIEW OF OPCODE STARTUP	9–57
CHAPTER 10 – MISCELLANEOUS SEQUENCERS	10–1
SECTION 10.1 – INTRODUCTION	10–1
SECTION 10.2 – INTRODUCTION TO KT, QA and QZ	10–5
SECTION 10.3 – THE KT SEQUENCER	10–6
SECTION 10.4 – THE QA SEQUENCER	10–13
SECTION 10.5 – THE QZ SEQUENCER	10–20
CHAPTER 11 – MEMORY SEQUENCERS	11–1
SECTION 11.1 – INTRODUCTION TO MEMORY CONTROL LOGIC	11–1
SECTION 11.2 – THE QM and QB SEQUENCERS	11–19

TABLE OF CONTENTS

PAGE

SECTION 11.3 – THE KP SEQUENCER 11–30

CHAPTER 12 – INPUT/OUTPUT SEQUENCERS 12–1

SECTION 12.1 – INPUT/OUTPUT CONTROL 12–1

SECTION 12.2 – THE KX SEQUENCER 12–19

SECTION 12.3 – QW OPERATING IN THE SHIFT–SEND MODE 12–24

SECTION 12.4 – BLOCK INPUT/OUTPUT 12–28

SECTION 12.5 – THE KM SEQUENCER 12–35

SECTION 12.6 – THE QC SEQUENCER 12–40

SECTION 12.7 – THE KW SEQUENCER 12–46

SECTION 12.8 – THE QW SEQUENCER 12–65

CHAPTER 13 – ADDER SEQUENCERS 13–1

SECTION 13.1 – INTRODUCTION 13–1

SECTION 13.2 – REPEAT OPERATIONS 13–12

SECTION 13.3 – KM SEQUENCER CONTROL OF REPEAT OPERATIONS 13–23

SECTION 13.4 – THE KJ SEQUENCER 13–29

SECTION 13.5 – THE KL SEQUENCER 13–46

SECTION 13.6 – THE QS SEQUENCER 13–54

SECTION 13.7 – THE KA SEQUENCER 13–85

SECTION 13.8 – INTRODUCTION TO MULTIPLY AND DIVIDE 13–111

SECTION 13.9 – KD SEQUENCER CONTROL OF MULTIPLY OPERATIONS 13–119

SECTION 13.10 – PRODUCT FORMATION BY THE QP SEQUENCER 13–129

SECTION 13.11 – KD SEQUENCER CONTROL OF DIVIDE AND REVERSE DIVIDE 13–149

SECTION 13.12 – QUOTIENT FORMATION BY THE QQ SEQUENCER 13–161

TABLE OF CONTENTS

PAGE

CHAPTER 14 – APPENDIX 14–1

APPENDIX I – REPEAT MODE BLOCK DIAGRAM 14–1

APPENDIX II – CD DECODING FOR GROUPS OF COMMANDS 14–5

APPENDIX III – CENTRAL PROCESSOR LOGIC SYMBOLS 14–7

APPENDIX IV – INDICATOR LIGHT CONNECTIONS FOR CENTRAL PROCESSOR LOGIC SYMBOLS 14–18

LIST OF ILLUSTRATIONS

FIGURE NO.	DESCRIPTION	PAGE
9.1–1	Lines of Communication Between Sequencers	9–2
9.3–1	Algorithm for Command Access	9–13
9.3–2	Algorithm for Operand Assembly; I: Command Word Requirements	9–15
9.3–3	Algorithm for Operand Assembly; II: K Level Sequencer Requirements	9–16
9.5–1	The KCA Section of Master Control	9–25
9.6–1	The KZC Section of Master Control	9–31
9.7–1	The KZE Section of Master Control	9–39
9.8–1	The KCL Section of Master Control	9–45
9.9–1	The KCS Section of Master Control	9–55
10.3–1	The KT Flow Chart	10–11
10.4–1	Algorithm for the Accumulator Put Function of QA	10–16
10.4–2	Algorithm for QA Manipulations for Multiply and Divide	10–17
10.4–3	The QA Flow Chart	10–19
10.5–1	Algorithm for Shifting the Operand to Zero Exponent	10–24
10.5–2	Algorithm for Setting Exponent Equal to Pickapoint Exponent for P1 (STS) Command Pickapoint Mode	10–25
10.5–3	The QZ Flow Chart	10–27
11.1–1	Basic Memory Timing Counter	11–5
11.1–2	Timing Diagram for Signals DAS, DAR and FC	11–16
11.2–1	Algorithm for the QM Sequencer	11–23
11.2–2	The QM Flow Chart	11–25
11.2–3	Algorithm for the QB Sequencer	11–27
11.2–4	The QB Flow Chart	11–29
11.3–1	Algorithm for Final Processing of Index Commands by KP	11–32
11.3–2	The KP Flow Chart for the Final Processing of Index Commands by K	11–35
11.3–3	Algorithm for the Processing of Store Logical and Store Zero Commands by KP	11–37
11.3–4	The KP Flow Chart for the Processing of Store Logical and Store Zero Opcodes	11–39
11.3–5	Algorithm for the Processing of Store Double Precision and Store Integer Opcodes by KP	11–43
11.3–6	The KP Flow Chart for the Processing of Store Double Precision and Store Integer Opcodes	11–45

LIST OF ILLUSTRATIONS

FIGURE NO.	DESCRIPTION	PAGE
11.3–7	Algorithm for the Processing of Store Single Precision Commands by KP	11–47
11.3–8	The KP Flow Chart for Store Single Precision Commands	11–49
11.3–9	The KP Flow Chart	11–53
12.2–1	Algorithm for the KX Sequencer	12–21
12.2–2	The KX Flow Chart	12–23
12.3–1	Algorithm for the Shift–Send Mode of QW	12–25
12.3–2	The QW Flow Chart	12–27
12.4–1	General Algorithm for Block Receive Operations	12–30
12.4–2	Block Receive Example	12–31
12.4–3	General Algorithm for Block Transmit Operations	12–33
12.4–4	Block Transmit Example	12–34
12.5–1	Algorithm for the Control of Block Input/Output Operations by KM	12–37
12.5–2	The KM Flow Chart for the Control of Block Input/Output	12–39
12.6–1	Algorithm for Accessing the Second Command Word for Repeat and Block Input/Output Operations	12–43
12.6–2	The QC Flow Chart	12–45
12.7–1	The KW Algorithm for Block Receive	12–49
12.7–2	The KW Flow Chart for Block Receive	12–53
12.7–3	The KW Algorithm for Block Transmit	12–55
12.7–4	The KW Flow Chart for Block Transmit	12–59
12.7–5	The KW Flow Chart	12–63
12.8–1	The QW Algorithm for Block Receive	12–67
12.8–2	The QW Flow Chart for Block Receive	12–69
12.8–3	The QW Algorithm for Block Transmit	12–71
12.8–4	The QW Flow Chart for Block Transmit	12–73
12.8–5	The QW Flow Chart	12–75
13.2–1	General Algorithm for Processing Repeat Commands Handled by KL	13–21
13.2–2	General Algorithm for Processing Repeat Commands Handled by KA	13–22
13.3–1	Algorithm for KM Control of Repeat Operations	13–25
13.3–2	The KM Flow Chart for Controlling Repeat Operations	13–27
13.3–3	The KM Flow Chart	13–28

LIST OF ILLUSTRATIONS

FIGURE NO.	DESCRIPTION	PAGE
13.4–1	Algorithm for the Processing of R2 and R3 Commands by KJ	13–33
13.4–2	The KJ Flow Chart for the Processing of R2 and R3 Commands	13–35
13.4–3	Algorithm for the Processing of X2 Commands by KJ	13–37
13.4–4	The KJ Flow Chart for the Processing of X2 Commands by KJ	13–39
13.4–5	Algorithm for Processing of the X3 Command and Enabled Interrupt Requests by KJ	13–41
13.4–6	The KJ Flow Chart for the Processing of X3 Commands and Enabled Interrupt Requests by KJ	13–43
13.4–7	The KJ Flow Chart	13–45
13.5–1	The KL Flow Chart	13–53
13.6–1	Summing Algorithm	13–57
13.6–2	The QS Flow Chart for Summing	13–61
13.6–3	Differencing Algorithm	13–65
13.6–4	The QS Flow Chart for Differencing	13–67
13.6–5	Basic Algorithm for Exponent Equalization	13–69
13.6–6	Algorithm for Equalizing Exponents when EL is High – (EA) > (EP)	13–70
13.6–7	The QS Flow Chart for Equalizing Exponents when EL is High – (EA) > (EP)	13–75
13.6–8	Algorithm for Equalizing Exponents when $\overline{\text{EL}}$ is High – (EA) < (EP)	13–77
13.6–9	The QS Flow Chart for Equalizing Exponents when $\overline{\text{EL}}$ is High – (EA) < (EP)	13–79
13.6–10	Miscellaneous Operations Performed by QS	13–81
13.6–11	The QS Flow Chart	13–83
13.7–1	Algorithm for the Processing of Non–Repeat Logic, Arithmetic and Address Preparation Opcodes by KA	13–93
13.7–2	The KA Flow Chart for Processing Non–Repeat Logic, Arithmetic and Address Preparation Opcodes	13–95
13.7–3	Algorithm for the Processing of Repeat Logic and Arithmetic Opcodes by KA	13–97
13.7–4	The KA Flow Chart for the Processing of Repeat Logic and Arithmetic Opcodes by KA	13–99

LIST OF ILLUSTRATIONS

FIGURE NO.	DESCRIPTION	PAGE
13.7–5	Algorithm for the Processing of Non–Repeat Logic and Arithmetic Tests Handled by KA	13–101
13.7–6	The KA Flow Chart for Processing Non–Repeat Logic and Arithmetic Tests	13–103
13.7–7	Algorithm for the Processing of Repeat Logic and Arithmetic Tests Handled by KA	13–105
13.7–8	The KA Flow Chart for the Processing of Repeat Logic and Arithmetic Tests Handled by KA	13–107
13.7–9	The KA Flow Chart	13–109
13.9–1	General Algorithm for Multiply Operations	13–120
13.9–2	Algorithm for KD Control of Multiply Operations	13–123
13.9–3	The KD Flow Chart for Control of Multiply Operations	13–127
13.10–1	Demonstration of Paths Between QPD and QPE	13–143
13.10–2	Algorithm for Product Formation by QP	13–145
13.10–3	The QP Flow Chart	13–147
13.11–1	General Algorithm for Divide and Reverse Divide Operations	13–150
13.11–2	Algorithm for KD Control of Divide and Reverse Divide Operations	13–153
13.11–3	The KD Flow Chart for Control of Divide and Reverse Divide Operations	13–157
13.11–4	The KD Flow Chart	13–159
13.12–1	Shift Paths Used in QQ	13–171
13.12–2	Algorithm for the Formation of 14 Octal Quotient by QQ	13–173
13.12–3	The QQ Flow Chart	13–175
14–1	Block Diagram of Repeat Mode	14–3

LIST OF TABLES

NUMBER	TITLE	PAGE
9.4–1	Blocks of KC Logic Used in Example	9–20
9.5–1	Terms Used on the Flow Chart of the KCA Section of Master Control	9–24
9.6–1	Terms Used on the Flow Chart of the KZC Section of Master Control	9–30
9.7–1	Terms Used on the Flow Chart of the KZE Section of Master Control	9–38
9.8–1	Terms Used on the Flow Chart of the KCL Section of Master Control	9–44
9.9–1	Basic Opcode Groupings	9–52
9.9–2	Gating Terms at KCT, KCX, KCZ, KZK	9–53
9.9–3	Groupings Used on the KCS Flow Chart	9–54
10.3–1	Decoding for Register Selection	10–7
10.3–2	Terms Used on the KT Flow Chart	10–10
10.4–1	Terms Used on the QA Flow Chart	10–18
10.5–1	Terms Used on the QZ Flow Chart	10–26
11.1–1	Memory Users	11–18
11.2–1	Terms Used on the QM Flow Chart	11–24
11.2–2	Terms Used on the QB Flow Chart	11–28
11.3–1	Terms Used on the KP Flow Chart for Final Processing of Index Commands	11–34
11.3–2	Terms Used on the KP Flow Chart for the Store Logical and Store Zero Commands	11–38
11.3–3	Terms Used on the KP Flow Chart for Store Double Precision and Store Integer Opcodes	11–44
11.3–4	Terms Used on the KP Flow Chart for Processing Store Single Precision Commands	11–48
11.3–5	Terms Used on the KP Flow Chart	11–52
12.1–1	Opcode in Second Command Word of Block Input/Output	12–10
12.1–2	Counting as Performed by the CP–CQ Counter	12–14
12.1–3	Input/Output Information	12–17
12.1–4	Input/Output Information	12–18
12.2–1	Terms Used in the KX Flow Chart	12–22
12.3–1	Terms Used on the QW Flow Chart for the Shift–Send Mode	12–26
12.5–1	Terms Used on the KM Flow Chart for Control of Block Input/Output	12–38
12.6–1	Terms Used on the QC Flow Chart	12–44
12.7–1	Terms Used on the KW Flow Chart for Block Receive	12–52

LIST OF TABLES

NUMBER	TITLE	PAGE
12.7–2	Terms Used on the KW Flow Chart for Block Transmit	12–58
12.7–3	Terms Used on the KW Flow Chart	12–62
12.8–1	Terms Used on the QW Flow Chart for Block Receive	12–68
12.8–2	Terms Used on the QW Flow Chart for Block Transmit	12–72
13.1–1	QS Inputs	13–7
13.1–2	Sign Determination	13–8
13.1–3	Comparison Between Sequencers That Use the Adder	13–10
13.1–4	Signals Important in Operation of the Adder	13–11
13.2–1	Processing Distinctions Between KA and KL	13–17
13.2–2	Processing of Arithmetic and Logic Commands	13–18
13.2–3	Processing of the Arithmetic Test and Logic Test Commands	13–19
13.3–1	Terms Used on KM Flow Chart for Control of Repeat Operations	13–26
13.4–1	Terms Used on the KJ Flow Chart for R2 and R3 Commands	13–34
13.4–2	Terms Used on the KJ Flow Chart for the X2 Command	13–38
13.4–3	Terms Used on the KJ Flow Chart for the X3 Command and Interrupts	13–42
13.4–4	Terms Used on the KJ Flow Chart	13–44
13.5–1	Use of the Adder by KL	13–49
13.5–2	Terms Used on the KL Flow Chart	13–52
13.6–1	Use of Exponents Generated by QS	13–55
13.6–2	Terms Used on the QS Flow Chart for Summing	13–60
13.6–3	Terms Used on the QS Flow Chart for Differencing	13–66
13.6–4	Paths Enabled During Exponent Equalization	13–72
13.6–5	Terms Used on QS Flow Chart for Exponent Equalization	13–74
13.6–6	Terms Used on QS Flow Chart for Exponent Equalization	13–78
13.6–7	Terms Used on the QS Flow Chart for Miscellaneous Operations Performed by QS	13–80
13.7–1	Comparison of KA Startups Through Use of Signal GKA	13–87
13.7–2	Terms Used on the KA Flow Chart for the Processing of Non–Repeat Logic, Arithmetic and Address Preparation Opcodes	13–94

LIST OF TABLES

NUMBER	TITLE	PAGE
13.7–3	Terms Used on the KA Flow Chart for the Processing of Repeat Logic and Arithmetic Opcodes	13–98
13.7–4	Terms Used on the KA Flow Chart for the Processing of Non–Repeat Logic and Arithmetic Tests	13–102
13.7–5	Terms Used on the KA Flow Chart for the Processing of Repeat Logic and Arithmetic Tests	13–106
13.7–6	Terms Used on the KA Flow Chart	13–108
13.8–1	Use of the Adder in Multiply and Divide Operations	13–114
13.8–2	Exponent Ranges and Associated Signals	13–118
13.9–1	Terms Used on the KD Flow Chart for Control Multiply	13–126
13.10–1	Partial Product Sign Values	13–132
13.10–2	Determination of Summing or Differencing Operations	13–138
13.10–3	Actions Taken Following Examination of the Least Significant Three Bits of the Multiplier	13–140
13.10–4	Paths in Product Formation States of QP: QPD and QPE	13–142
13.10–5	Terms Used on the QP Flow Chart	13–146
13.11–1	Terms Used on the KD Flow Chart for Controlling Divide and Reverse Divide Operations	13–156
13.12–1	Actions Taken on the Basis of the Remainder Sign and Carry Signal During the Operation of QQ	13–170
13.12–2	Terms Used on the QQ Flow Chart	13–174

CHAPTER 9

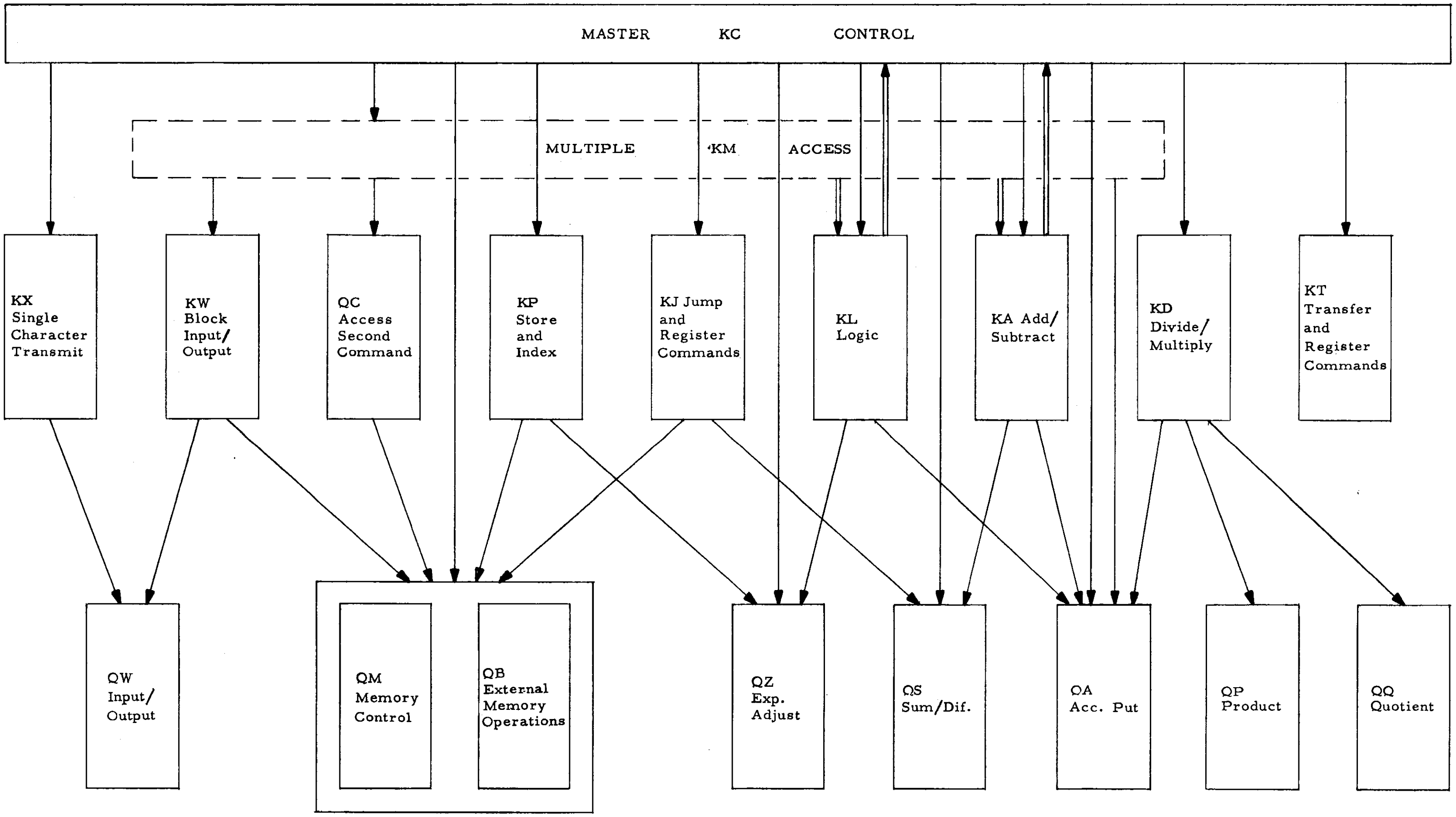
MASTER CONTROL

SECTION 9.1 – INTRODUCTION

The G–20 operation codes are performed by a single complex of circuitry. For purposes of design and discussion, the functions handled by this circuitry are divided into sections called sequencers. By this means functions are easily identified and discussion is limited to a reasonably small set of manipulations. It should be emphasized that this organization is strictly for referencing convenience. Neither the discussions of individual sequencers in this manual nor the depiction of each on a separate flow chart should be taken as an indication that each has unique circuitry; on the contrary, circuitry is shared wherever possible.

Since the tasks performed by the sequencers vary in importance, a hierarchy has been established. The lowest level sequencers are those which perform bookkeeping–type operations. These are called building block sequencers, designated by Q; each handles one or more basic tasks. On a higher level are the K sequencers, called opcode level sequencers because each handles the final steps in the processing of a group of operation codes. The K level sequencers use the building block sequencers – and, in the case of repeat operations, the Master Control sequencer – to carry out specific portions of their assigned tasks. Over–all control is effected by the Master Control sequencer,

FIGURE 9.1–1 Lines of Communication Between Sequencers



designated KC. Despite the use of the name, KC, Master Control is not a K level sequencer; it comprises the step above the K level sequencers. Thus, where the term "K level sequencers" appears in this write—up, it is understood that KC is not included. Master Control uses the Q level sequencers, where appropriate, and starts up the K level sequencers when the final operand is available in the required form and all preliminary manipulations are completed. This operation is referred to as opcode startup. For some non—complex opcodes which use only one operand, KC does not start a K level sequencer. Figure 9. 1—1 indicates the possible lines of communication between sequencers. (Double lines are used to distinguish repeat operations.)

The relationships that exist between sequencers require a certain amount of explanation. Confusion arises from control transfers, the simultaneous operation of two or more sequencers, loops within sequencers, etc. Because the sequencers are so interdependent, a description of any one of them must include information concerning the environment. The interdependent and repetitive aspects of the sequencers create confusion in following execution of an opcode on the sequencer flow charts. The basic operations performed by each sequencer are quite simple; the manipulations involved in adapting all cases to these operations make it difficult to follow the basic operations. This is particularly true in the case of Master Control where provisions are included for all possible combinations of mode, index, and opcode.

Description of Master Control presents an additional difficulty in that, as the initiator of all activities, it is responsible for instructing itself. The task currently being executed by KC is always predicated on the previous one with the chain of events going back to the time the machine was bootstrapped. (The bootstrap operation, described in the section on input/output, allows for the initialization of the Central Processor.) Thus, in each sequencer write—up, considerable emphasis is placed on the relation of this sequencer to the others with which it communicates. This should make it possible to grasp the G–20 logic as an integrated whole rather than as a series of disconnected fragments.

Basic documentation for the G–20 logic sequencers are the 19 sequencer flow charts. These are time oriented, showing what happens on each clock within each sequencer from the time it is taken from its idle loop to the return to this loop. Each clock time is called a sequencer state. These states are referred to by their letter names, using the name of the sequencer and sequential letters in the alphabet. Thus, the KT sequencer has states KTA, KTB, KTC, KTD, KTE and KTF. (Letters I and O are omitted to avoid confusion.) The A–B states of each sequencer represent the idle loop in which it remains until its start signal arrives. When the machine is initialized, most sequencers are sent to their A–B states. Those that are used for receiving the bootstrap program are set to the states appropriate for handling this function.

Sequencer states are distinguished from one another by means of flip–flops which, again, use the name of the sequencer, this time associated with numbers. In short sequencers, four flip–flops are used: two on clock 1, two on clock 2. In the longer sequencer, eight flip–flops are necessary to provide enough distinct states. KC is a long sequencer and thus uses eight flip–flops. Here, the KC1 flip–flop on clock 1 is echoed by KC2 on clock 2: KC3 on clock 1 is echoed by KC4 on clock 2. This state information is shown in the left hand margin of the chart as follows:

KCA which represents $C1 \wedge \overline{KC7} \wedge \overline{KC5} \wedge \overline{KC3} \wedge \overline{KC1}$
0000

KCB which represents $C2 \wedge \overline{KC8} \wedge \overline{KC6} \wedge \overline{KC4} \wedge \overline{KC2}$
0000

KCC which represents $C1 \wedge \overline{KC7} \wedge KC5 \wedge \overline{KC3} \wedge \overline{KC1}$
0100

KCD which represents $C2 \wedge \overline{KC8} \wedge KC6 \wedge \overline{KC4} \wedge \overline{KC2}$
0100

etc.

No set pattern is used for the changing of states; sometimes 11 follows 10, sometimes not. Sequencers skip in their states by means of changes in the state flip–flops. Thus, to get from

QSF (which represents $C2 \wedge \overline{QS6} \wedge QS4 \wedge QS2$)
011

to

QSL (which represents $C1 \wedge QS5 \wedge \overline{QS3} \wedge QS1$)
101

it is necessary to set QS5 and reset QS3. Since C2 flip–flops echo C1, no jumps in the logic occur between C1 and C2; they always follow C2. The flow charts show the decisions and resulting branches made at each sequencer state. The decisions are based on the signals read at that clock time. These signals are called the gating terms and are shown above the boxes. Inside the boxes are shown the events that occur when the particular branch is taken. The setting and resetting of sequencer state flip–flops have not been included in the boxes since it is clear from a glance at the state decoding that must be set or reset in order to get to the next state. Also omitted from the boxes are redundant terms which occur when one branch at a given state requires enabling of a path or changing of a flip–flop state while another branch does not; if this term does not in any way affect the other branch, it is not inhibited on the non–affected branch (since more logic would be required to do so) and arrival at this particular state will cause this event to occur. These terms have been left off the branches they do not affect because they tend to obscure what is happening, and, in fact, can cause a good deal of confusion if a reason for this event is sought.

A third omission that should be pointed out is that of an indication of which flip–flop is being read for the flip–flop signals shown: the clock 1 or clock 2. This has been done in the same spirit of economy, since there are a considerable number of terms included and the addition of 1 or 2 to the end of each makes them harder to read. It should be understood that for clock 1 states (QS$\underline{A}$, QS$\underline{C}$, QS$\underline{E}$) the flip–flops which are stable on clock 1, e.g., SZ1, SM1, will be read, while on clock 2 the opposite case exists, e.g., flip–flop SZ2 and SM2 are stable. Also, if a path is enabled on clock 2 to copy the state of one flip–flop into another, as, for example, the path AS(0)WS, it is assumed that the AS flip–flop is being read on clock 2. This means that AS2 is stable and is being used to affect the state of WS1 so that it will reflect the state of AS2 on the next clock 1. Hence, the enabled path is AS2(0)WS1. Similarly, the path WS(C)SM, enabled on clock 2 causes the SM1 flip–flop to be stable on clock 1 in the opposite (or complement) state from that of the WS2 flip–flop.

A few words of introduction concerning terminology may save the reader a great deal of time; it is always easier to remember initials if it is understood what they represent. The names of the 19 sequencers have the following origins:

KA	–	ARITHMETIC operations;
KC	–	CONTROL K level sequencers;
KD	–	DIVIDE/multiply;
KJ	–	JUMP and register commands;
KL	–	LOGIC operations;
KM	–	MULTIPLE access;
KP	–	PUTAWAY and index commands; (store commands were originally called putaway);
KT	–	TRANSFER and register commands;
KW	–	Block input/output (transmit/receive);

KX – TRANSMIT single character (from Xmit);

QA – ACCUMULATOR put (store in Accumulator);

QB – Memory control – B (operates in conjunction with QM when external memory is used);

QC – Access second COMMAND word in multiple access operations;

QM – MEMORY control;

QP – PRODUCT (multiply);

QQ – QUOTIENT (divide);

QS – SUM/difference operations;

QW – Block input/output (transmit/receive);

QZ – Shift operand to ZERO exponent or pickapoint exponent.

It should also be understood that the general term "Accumulator" is used to refer to the A register, the AE exponent register, and the AS sign flip–flop; thus, if an operand is stored in the Accumulator, information is sent to A, AE, and AS. The term, Accumulator, is often abbreviated to Acc. Similarly, a reference to OA implies use of register N, exponent register EA, and sign flip–flop WS. A corresponding operand storer is the D, EP, SM complex, but this has been given no special name, probably because the programmers do not need to refer to it. In keeping with common usage, parentheses indicate "the contents of". Thus, "(D) are sent to register S" is read "the contents of D are sent to register S".

SECTION 9.2 – PROCESSING OF A SIMPLE OPCODE

The activities of Master Control are surprisingly repetitive. The same operation is often performed several times at various stages in the processing of a single command. For example, consider the steps necessary in processing the command "clear and add to the Accumulator the contents of the location specified by the addition of the contents of index register 37_8 to the contents of location 362_8". Execution proceeds as follows: (1) the command word is accessed and decoded, (2) the operand is assembled, (3) since this is mode 3 and asks for the contents of the assembled operand as the final operand, the former is used as an address for the final access, and (4) the result of this final access is stored in the Accumulator.

In this example, two of the three tasks performed by KC are demonstrated; command access and operand assembly. The command is executed entirely by KC so that no opcode startup is necessary.

Assume these initial conditions: completion of the preceding opcode has been signaled, the address of the next command (stored in register CA) has been incremented, the previous command left nothing useful in OA, and the following octal values are relevant:

(OA) = 0
(Register CA) = 01102
(Location 01102) = 06053700362
(Location 00362) = 00000000032
(Location 00037) = 00000000016 (This is an index register)
(Location 00050) = 00000000236

Master Control begins by accessing the memory location designated by the address stored in register CA. This brings the command word

06053700362

into register B. Bits 29—15 of this command are sent to the command decoding circuitry. If bits 14—0 contain a number destined for use as part of the operand (modes 0 or 2) the number is sent to the Arithmetic Unit where it is added to the contents of OA. If, instead, it is an address (such is the case in this example) it is sent to register BA to be used as the address for the next access. Specifically, the address 00362 is sent to BA. When this location is accessed

00000000032

is left in register B. It should be noted that this memory access does not differ from the earlier one. When the second access occurs, however, Master Control has advanced in its logic so that this information will not be decoded as a command word. Instead, it will be sent to the Arithmetic Unit to be added to the contents of OA.

Next, the index address from the command word is sent to the BA register and that location is accessed. This brings

00000000016

into register B and this quantity is also added into the operand. The result of this addition (the number 50_8) is to be used as the address of the final operand, thus necessitating an additional memory access which brings the contents of location 50

00000000236

into register B. The final step in the operation is the storing of the mantissa in the Accumulator register and its exponent (in this case, zero) in the Accumulator exponent register.

Despite the relative simplicity of the example, four memory accesses and three addition operations were necessary. Since memory accesses are accomplished by the QM sequencer and add/subtract operations by QS, this meant startup of QM four times and QS three times. If the processing of this operation were to be explained step by step on the KC flow charts, the discussion would include the relevant events at each of the following states (this is assuming that all numbers accessed are single precision): KCA, KCB, KCG, KCH, KCJ, KCK, KCL, KCM, KCS, KCT, KZE, KZF, KCL, KCM, KCS, KCT, KZC, KZD, KCL, KCM, KCS, KCT, KCA. Confusion is bound to result from a description of this kind, but the principal objection to this approach is that the reader will not gain an understanding of the sequencer activities. He will only be able to follow each step involved in this particular operation. In view of the fact that there are 104 G–20 operation codes, and that each can be modified through use of modes and most of them through the use of indices, the probability is high that he would be unable to follow a second operation without prompting. Examination of individual cases does not serve as a good introduction to Master Control; thorough understanding of this sequencer can be attained only through analysis of its logic.

SECTION 9.3 – TASKS PERFORMED BY KC

The three principal tasks carried out by KC are the accessing of commands, the assembly of operands, and the startup of the appropriate opcodes. The handling of each of these functions is described in general terms in this section. A more specific discussion of the logic appears later in the chapter. In general, it can be said that the paths taken in KC are determined by the status of the program being executed (i.e., is the next access a normal access or has an interrupt been requested, etc.), the precision of the information being accessed, the mode and index information contained in the command word, the operand format requirements for the particular opcode, the timing considerations which result from simultaneous operation of KC with other sequencers, and the requirements of the opcode itself.

<u>9.3–1 COMMAND ACCESS</u> The sequencer logic assumes that the commands of programs to be executed are stored consecutively unless otherwise indicated, i.e., unless the program specifically calls for a jump to another location. When execution of any program begins, the starting location is sent to register CA. This number is normally incremented by 1 each time a command is executed. If a command calls for a jump to, say, location ALPHA, ALPHA is sent to CA. If the command is a transfer and mark to location ALPHA, the address of the current command plus 1 is stored in location ALPHA, and ALPHA plus 1 is sent to CA.

When the branch in the program has been executed, the program will call for a jump to the contents of location ALPHA which will cause the address stored in ALPHA to be sent to CA. The handling of these addresses and the incrementing of CA are carried out by the lower level sequencers which complete processing of the opcodes. These sequencers also inform KC of the opcode completion. KC is then responsible for accessing the command whose address is currently stored in CA, assembling the designated operand, and sending a start signal to a K level sequencer if required.

Command access proceeds on the basis of the following:

(1) accessing of a new command begins when KC is signaled that the last operation is completed or when an early start signal is received indicating that it is safe to proceed with the next access even though the current operation is not entirely finished;

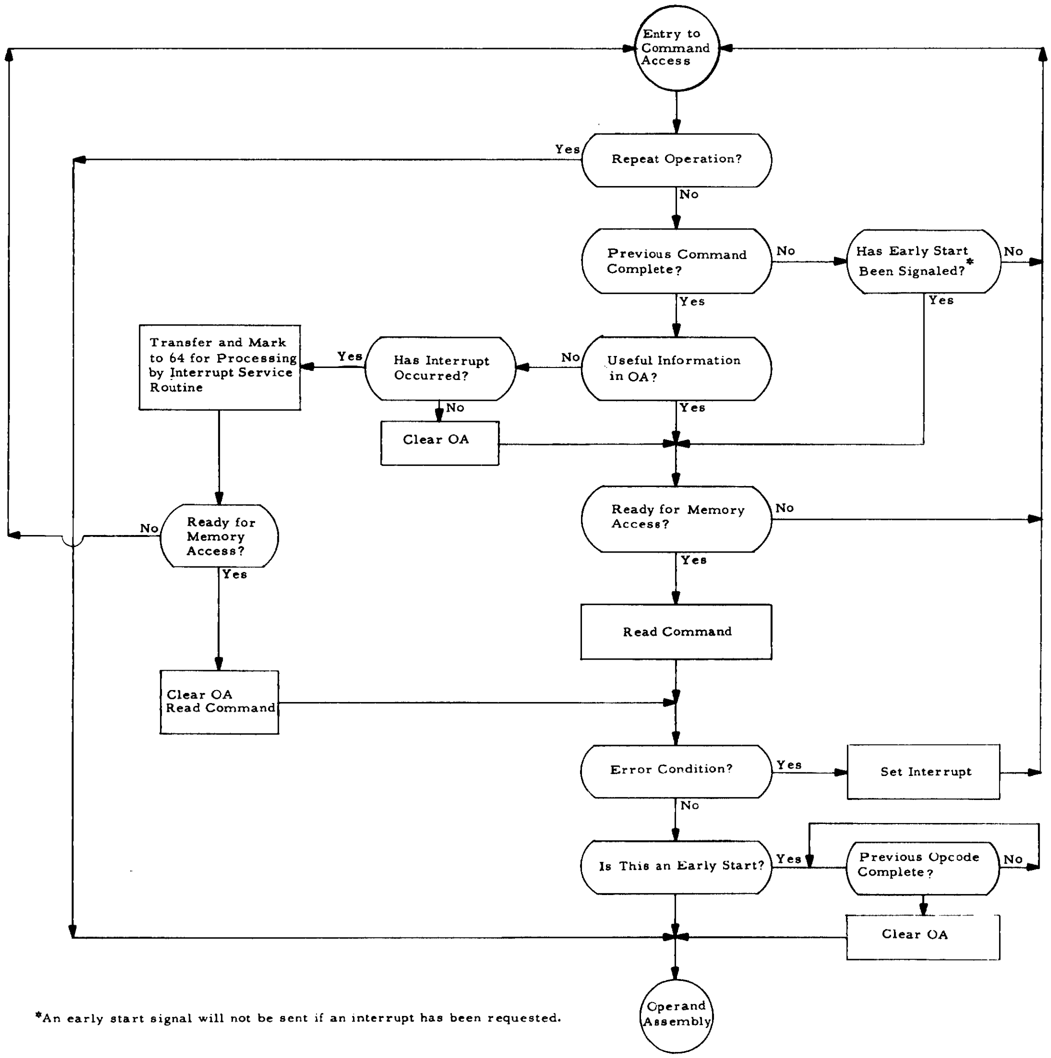
(2) if a repeat operation is in progress, the command access logic is skipped (this happens because the same command is being repeated for a block of addresses);

(3) if the previous command was an OA command (leaving information in the OA register so that it will affect the operand assembled for the following command) the OA register is not cleared; otherwise, it is cleared so that any information it contains will not affect the formation of the new operand;

(4) if an interrupt has been requested, a transfer and mark to location 64 is effected. This causes entry to the Interrupt Service Routine where action appropriate to the particular interrupt is taken. In order to prevent the possible loss of information, interrupts are not processed until the current opcode is completed and unless the previous command was something other than an OA command. In the case of repeat operations, no external interrupts are processed. The operation will be terminated by the occurrence of an internal interrupt whether due to an error condition or the occurrence of a flag.

The general algorithm for command access appears in Figure 9.3-1. The flow chart for this section of the logic is discussed in Section 9.5 while the flow chart appears in Figure 9.5-1.

FIGURE 9.3-1 Algorithm for Command Access



*An early start signal will not be sent if an interrupt has been requested.

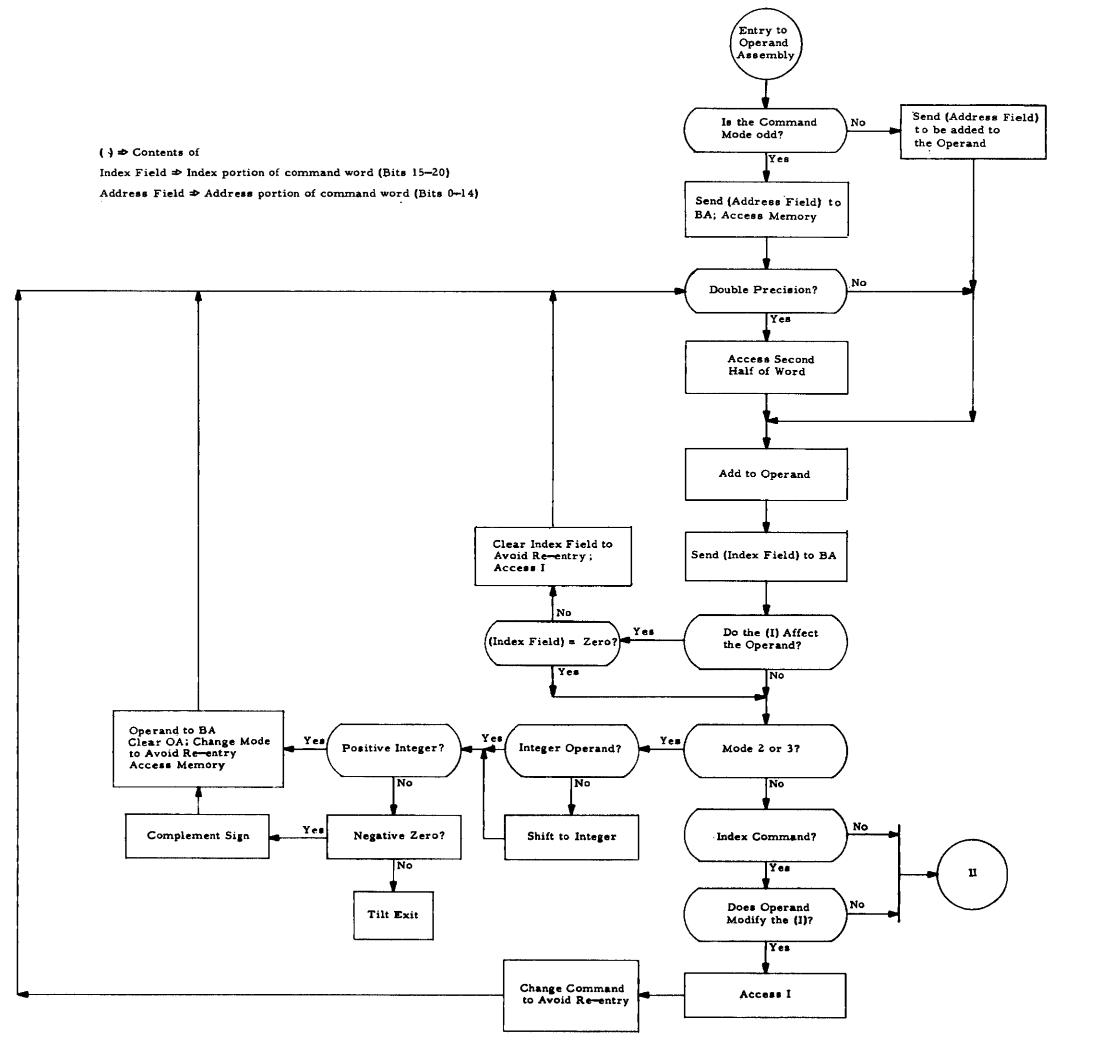
9.3-2 OPERAND ASSEMBLY The process of operand assembly follows the algorithm shown in Figure 9.3-2 and 9.3-3. Examination of these charts will demonstrate how provisions have been made for any legitimate opcode modification. It should be noted that branchings for repeat operations are not indicated on the flow chart since all such branchings (with a single exception at KCS) are for the purpose of indicating that the address of the next word to be accessed is stored in register CA rather than in register BA. This is due to a peculiarity of repeat operations and does not in any way affect the general rule for operand assembly. The branch at KCS is discussed later.

The flow chart is divided into two parts: I, actions determined by the requirements of the command word, i.e., the mode, index information, requirements of the opcode, etc., and II, requirements of the K level sequencer that will complete processing of the particular opcode. Handling of the indicated decisions is discussed in detail in Sections 9.6 and 9.9.

9.3–3 OPCODE STARTUP This operation consists of sending a start signal to the K level sequencer responsible for final processing of the current command. In following the execution of a given command on the flow charts it should be remembered that, for most opcodes, startup may occur at more than one state of KC depending upon how soon the operand conforms to the operand requirements of the particular opcode. For those opcodes which impose no restrictions on the operand, only one startup from KC is possible. Others may be started from 2, 3, and in some cases, 4 different places. Alternate starts are included so that time will not be lost if the operand meets the requirements without being manipulated, while provisions are made for startup from each point at which it is possible that the requirements have been met.

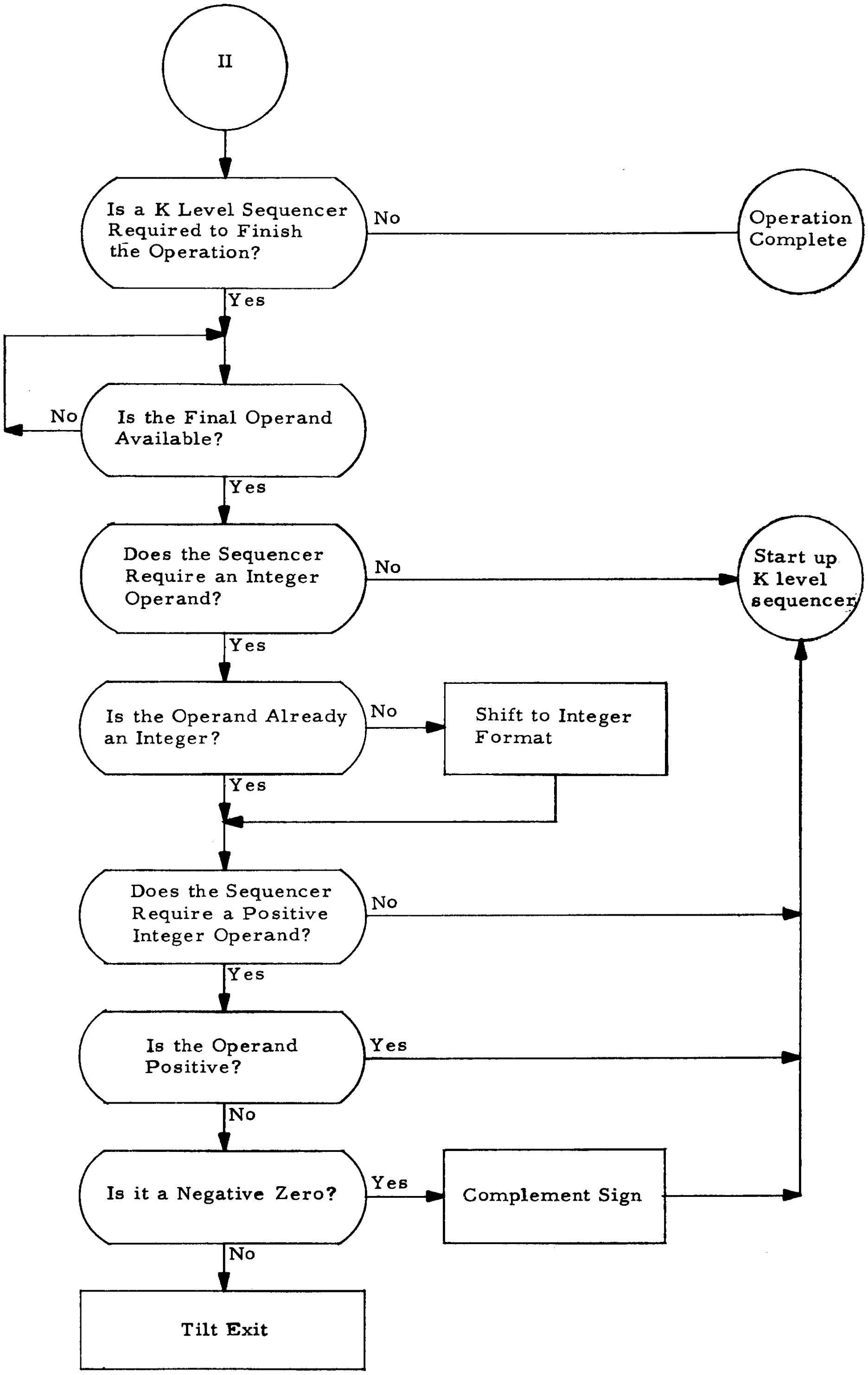
One unusual case of opcode startup exists: the KA sequencer may be started early when the commands it handles are written in mode 2 or 3.

FIGURE 9.3—2 Algorithm for Operand Assembly; I: Command Word Requirements


( ) ⇒ Contents of
Index Field ⇒ Index portion of command word (Bits 15—20)
Address Field ⇒ Address portion of command word (Bits 0—14)
Entry to Operand Assembly
Is the Command Mode odd?
No
Send (Address Field) to be added to the Operand
Yes
Send (Address Field) to BA; Access Memory
Double Precision?
No
Yes
Access Second Half of Word
Add to Operand
Send (Index Field) to BA
Clear Index Field to Avoid Re—entry; Access I
No
(Index Field) = Zero?
Yes
Do the (I) Affect the Operand?
Yes
No
Operand to BA Clear OA; Change Mode to Avoid Re—entry Access Memory
Yes
Positive Integer?
Yes
Integer Operand?
Yes
Mode 2 or 3?
No
No
No
Complement Sign
Yes
Negative Zero?
Shift to Integer
Index Command?
No
No
Yes
II
Tilt Exit
Does Operand Modify the (I)?
No
Yes
Change Command to Avoid Re—entry
Access I

This is possible because the initial operations performed by KA do not conflict with or depend upon those still going on in KC.

FIGURE 9.3-3 Algorithm for Operand Assembly; II: K Level Sequencer Requirements


II
Is a K Level Sequencer Required to Finish the Operation?
No
Operation Complete
Yes
No
Is the Final Operand Available?
Yes
Does the Sequencer Require an Integer Operand?
No
Start up K level sequencer
Yes
Is the Operand Already an Integer?
No
Shift to Integer Format
Yes
Does the Sequencer Require a Positive Integer Operand?
No
Yes
Is the Operand Positive?
Yes
No
Is it a Negative Zero?
Yes
Complement Sign
No
Tilt Exit

SECTION 9.4 – BREAKDOWN WITHIN THE LOGIC

KC is easily separated into four sections on the basis of the four transfer states that occur in the logic: KCA, KCL, KZC, KZE. For purposes of discussion, the KCL section, which is by far the most complex, is further subdivided into KCL and KCS. The approach to KC taken in the succeeding paragraphs attempts to clarify the logic of the entire sequencer through presentation of a detailed analysis of each of the sections in conjunction with explanations of how each implements the KC tasks discussed earlier.

Understanding of KC also requires knowledge of QS, QZ, QA and QM, the building block sequencers used by KC as subsections of its own logic. KC remains in control during the operation of these Q level sequencers and continues to advance in its own logic while the building block sequencers complete their tasks. Since processing is simultaneous, and KC is sometimes held up while the Q level sequencers complete their tasks, an understanding of these Q level sequencer functions and the signals they generate on completion is essential to an understanding of KC.

(1) The QM sequencer controls memory accesses and stores. No memory operation can be started until the (RQM READY) QM signal is available. QM is started by a GCA or a GBA signal, depending upon whether the address of the location to be accessed is stored in register BA or register CA. Memory operations also require sending of an MS or Memory Start signal and, in addition, it is necessary to clear the B register prior to starting the memory because the transfer into register B is true sides only. Thus, memory accesses are indicated on the flow chart as "Clear B, MS, GCA". When the addressed information is available in register B, a DAS

(Data Available Signal) is sent by QM.

(2) The QZ sequencer is used by KC to shift the operand to integer format (zero exponent). QZ is started by means of a GQZ (GO QZ) signal and sends a ZE2 (zero exponent) signal when the final operand is available.

(3) The QA sequencer is used by KC to store the result of an operation in the Accumulator (registers A and AE, sign flip–flops AS). It is started by GQA and sends a KR signal to KC after it has completed all but its bookkeeping operations so that no unnecessary time is lost before accessing of the next command begins.

(4) QS is the most important of the building blocks with respect to involvement with the logic of KC. QS adds the contents of register N to the contents of register D and gates the result into register S. The following signals, necessary to KC, are generated by QS:

1) DQS – DONE QS; DQS is high when the sequencer has completed the current operation and the result is available in OA. (Note that DQS remains high for only one clock.)

2) RQS – READY QS; RQS is high any time QS is not in operation.

3) ZE – Zero Exponent; ZE is high when the operand resulting from the addition has a zero exponent and the operation is complete. (Note that two terms are necessary to indicate a non–zero exponent, i.e., $\overline{\overline{ZE}} \wedge$ DQS.)

4) WS – Working Sign flip–flop; WS holds the sign of the

operand stored in OA; WS high indicates a negative sign.

A WG1 signal followed by GQS $\wedge$ WG2 are used by KC to start the QS sequencer. These signals are sent at the KCL and KCM or the KCQ and KCR states, depending upon whether the information being accessed is single or double precision. In either case, the first call for DQS (at KCT) occurs only one clock time after QS is started. QS has provisions for a quick exit in the case of a zero operand in register N. Thus, DQS can be high at KCT only if (N) = 0. When (N) ≠ 0, DQS comes high at KCZ. (An idle loop between KCW and KCX allows time for QS to finish.)

Similar considerations explain the manner in which exponents are handled at the KCT and KCX states. In the case where the (N) are initially zero, the QS sequencer leaves the exponent of the result in registers EP and ES. Thus, in order to update the EA register, which may contain spurious information, KC transfers the exponent via the ES(0)EA path.

If QS does not finish early, i.e., if (N) were other than zero, QS leaves the exponent in register EA and the exponent plus one is in register ES. Thus, the uninformed register at KCZ is EP. KC updates EP by means of enabling the ES(−1)EP transfer path. Consequently, at KCT the ES(0)EA path is enabled while at KCX, the path is EA(−1)EP.

The KCZ state is entered only when the QZ sequencer has been started for the purpose of shifting the operand to zero exponent. In this case no exponent manipulations are required beyond the clearing of the EP register since QZ will shift the operand until (EA) = (EP).

Each of these sequencers is discussed in detail later in the manual. These remarks are intended to orient the reader and to enable him to grasp the way in which KC depends upon these Q level sequencers for specific operations. It was pointed out earlier that KC used the QM and QS sequencers several times in performance of the opcode used as an example. It might also be noted that some logic sections of KC itself are used more than once in performance of this operation. Analyzed according to use of the five blocks of logic, the example can be diagrammed as follows:

TABLE 9.4–1 Blocks of KC Logic Used in Example

Entry Points	Time →									
KCA	X									X
KCL		X			X			X		
KCS			X			X			X	
KZE				X						
KZC							X			

This illustrates some of the characteristics of the KC logic: the KZE and KZC blocks can be entered only once during an operation, an operation is comprised of all events between advancement from KCA to re–entry at KCA (with the exception of repeat operations which enter at KCA on each iteration), and KCL and KCS may be entered several times (the same number of times each since progress is always from KCL to KCS). The same questions are asked each time a state is entered, but the decision and, hence, the paths followed, may be different each time.

One term which occurs throughout KC requires a bit of explanation: signal RKM (READY KM sequencer). The KM sequencer is used only for multiple access operations (block input/output and repeat arithmetic and logic commands). RKM is always high except when KM is in operation. Thus, $\overline{\text{RKM}}$ indicates repeat operation. Branching on this term occurs throughout KC when memory accesses are required because the address to be accessed is in register BA for non-repeat operations and in register CA for repeat. This necessitates the sending of a GCA command in one case, GBA in the other. At KCS, a similar branch occurs but for a different reason. In this case, the use of the BA register is involved. All repeat operations call for storage of the complemented block length (which specifies the number of times the operation is to be repeated) in register BA. However, for non-repeat operations, entry at KCS calls for the transfer of the index portion of the command word (which can only contain an index address) to register BA where it will be used by QM in accessing the index information. The branch at KCS prevents this transfer during repeat operations and thus protects the block length in BA.

SECTION 9.5 – ENTRY AT KCA

This is the most straightforward section of logic: the commands are accessed, the mode, index, and opcode information is sent to register CD, and decoding begins. Thus, all of the command access logic is included as well as the initial steps of operand assembly.

The general algorithm for command access is shown in Figure 9.3–1. Command access occurs when the previous command is complete, or an early start signal has been sent, and the memory is available. Four actions can be initiated at the KCB state:

(1) accessing of the next command,

(2) processing of an interrupt,

(3) a jump to the KCJ state if this is a repeat operation,

(4) return to the idle loop while waiting for RQM or KR2 or SKC.

The occurrence of an interrupt request takes precedence unless either a repeat operation is in progress or the previous command was an OA command. The programmer is allowed the flexibility of storing information in OA through use of OA commands in order to modify the operand assembled for the next command. To protect this information, any time an OA command is used the command which follows it must be processed immediately. The intervention of an interrupt would result in loss of information. During repeat operations, external interrupts are not recognized. Internal interrupts, whether due to flags or error conditions, terminate the operation and the interrupt is processed if Master Interrupt Control is on.

When the command has been accessed and is available in the B register, the bits containing mode, opcode and index information (29B15) are sent to the CD register. If the number in the address field (14B0) is to be used as an address (i.e., if this command is written in an odd mode, 1 or 3), the address is sent immediately to the BA register; if not, it is sent to the Arithmetic Unit to be added into the operand. This addition is accomplished by means of the QS sequencer which adds this new value to the contents of OA. Bits 30 and 31 of register B are checked for command flags; an interrupt will occur when processing of the command is completed if the command word was flagged and the appropriate bit position in register U was set. (This is referred to on the flow chart as "enable JCA, JCB".)

The processing of interrupts begins with the sending of a start signal, GKJ, to the KJ sequencer. KJ then performs the desired operation, a transfer and mark (command X3) to location 64_{10} (100_8), the address that marks the beginning of the Interrupt Service Routine. Since the sending of 64 to CA is contrary to the usual procedure for the X3 opcode (the destination of the transfer can ordinarily be any desired location in the program) a modification must be effected. This is handled by decoding of the first KC interrupt state, KCE, which causes KJ to follow the path wherein 64 is sent to CA. This state of affairs is indicated on the KC flow chart by the notation KJH ⇒ 64(0)CA.

Thus, KJ processes the interrupt cases as it would an X3 command with the exception the 64, rather than the assembled operand, is sent to CA and no interrupts can be requested during the processing.

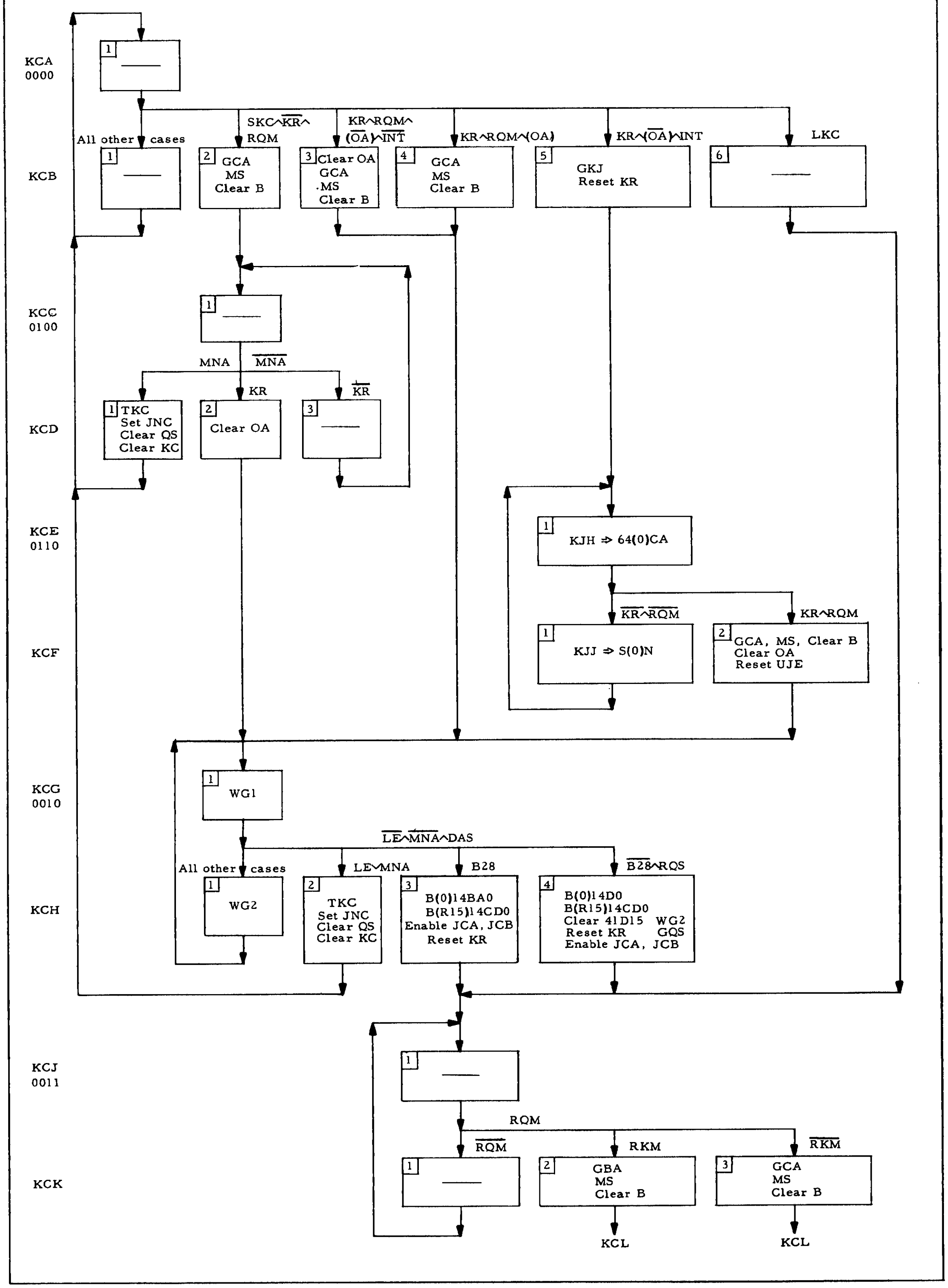
Figure 9.5–1 shows the KCA section of Master Control. The terms used on the flow chart are defined on the opposite page.

TABLE 9.5–1 Terms Used on the Flow Chart of the KCA Section of Master Control

B(R15)14CD0	Path enabled to send the index and command information from the command word to the command decoding register.
B28	Bit 28 in the B register; this flip–flop will hold mode information when a command word has been accessed; thus, B28 high indicates odd mode, B28 low, even mode.
B(0)14BA0	Path enabled for mode 1 or 3 commands to send address information from the command word to register BA where it will be used bv the QM sequencer in accessing the contents of the addressed location.
B(0)14D0	Path enabled for mode 0 and 2 command words to send operand information from the command word to register D where it will be added directly into the operand. The remaining bits in register D are cleared via the enabling of the Clear 41D15 path.
Clear B	Clear register B; information read from memory is sent to the B register; since this is a one–sided transfer, it is necessary to clear B before performing the access.
Clear OA	Clear Operand Assembly information; this involves the following operations: clear register N, clear exponent register EA, reset the WS flip–flop, set the SM flip–flop.
Enable JCA, JCB	Enable detection of command flags.
DAS	Data Available Signal from QM; DAS is sent when the accessed information is available in the B register.
GBA, GCA	Signals that start the QM sequencer; GCA is sent when the address is held in register CA; GBA is sent when the address is BA.
GQS	GO signal to QS sequencer.
Int	Indicates that an enabled interrupt has occurred.
KR	Opcode DONE signal to KC; KR is reset at KCH so that it can be used at the completion of the current opcode.
LE	Large Exponent signal; LE is sent by the QS sequencer when an exponent overflow occurs during its operation; QS hangs up in an idle loop until cleared by higher control; KC requests an interrupt, clears QS, and returns to idle.
LKC	Loop KC sequencer; LKC is sent by the KM, KA or KL sequencers during the processing of repeat operations to direct the access of the next operand.
MNA	Memory Non–existent signal; MNA is high when an illegal address is detected during operation of the QM sequencer; KC clears QS, requests an interrupt and returns to idle.
MS	Memory Start signal necessary to the initiation of any memory operation.
(OA)	Useful information has been stored in OA via use of an OA command; in this case OA is not cleared prior to operand assembly; $(\overline{OA})$ indicates that the information left in OA is not relevant and that OA should be cleared.
RQM	READY QM sequencer; no memory operation can begin until this signal is high.
SKC	Early START signal to Master Control; SKC gates the access of the next command word before the proceeding operation is completed.
TKC	Tilt KC sequencer; this signal is high when KC exits from the normal processing paths due to the occurrence of an error condition which results in an interrupt request.
UJE	Master interrupt control enable flip–flop; UJE is reset at KCF to inhibit the processing of an interrupt during the operation of the interrupt service routine which has just been entered via the transfer of 64_{10} to the CA register.*
WG1	Wait Gate signal; WG1 is a signal to the QS idle loop to enable the N(L3)S.
WG2	Wait Gate signal; WG2 is not the echo of WG1; WG2 is necessary in conjunction with GQS in order to start the QS sequencer.

FIGURE 9.5–1 The KCA Section of Master Control

KC



SECTION 9.6 – ENTRY AT KZC

Entry at KZC occurs under the following conditions: the relevant information in the A and I fields has been assembled; the command is written in mode 2 or 3 indicating that this assembled operand is to be used as an address in accessing the final operand; this address is in integer format.

If the address is positive and the QM sequencer is ready, the following actions occur:

(1) register B is cleared, memory access is started, and the decoding of the mode of the command is changed, by the resetting of CD14, to 0 or 1 (depending upon whether the original mode was odd or even) so that KZC will not be entered again;

(2) the KA sequencer is started if the command is any arithmetic operation or test, any OA command other than OCA or OCS, or one of the logic commands requiring arithmetic operations (L2, L3, S2, S3);

(3) LP is set for all logic operations and tests that are handled by the KL sequencer. (This excludes the four listed above that are handled by KA.) It should be noted that the final operands for all mode 2 or 3 logic commands are accessed in logic format. (Formatting is discussed in Section 9.8);

(4) for those commands which are not handled by KA, the notation "Clear OA" appears. (Clear OA = clear N, clear EA, reset WS, set SM.) This means that previous information stored in these registers and flip–flops will not affect the final operand when it is accessed;

(5) Four of the commands that call for the startup of KA are single operand commands (2f: A0, A1, T0, T1). For these commands, the N register is cleared so that the accessed operand will be added to zero when the QS sequencer is started. (Register EA is not cleared since the contents of EA must equal zero before the KZC loop is entered.) For the two operand commands which involve KA, the contents of register N, which have already been sent to the BA register for use as the address of the final operand, are destroyed by the superpositioning of the contents of the Accumulator. The transfers involved here are A(0)N, AE(0)EA, and AS(0)WS. Thus, when the final operand is accessed and stored in registers D and EP, the startup of QS will perform the sum or difference operation required on the two operands. The early startup of KA involves manipulations of the sign values attached to the operands and to the commands itself. Since sign values are basic to all arithmetic operations, the detailed discussion of their handling has been left to Chapter 13 which includes all the sequencers that use the Adder. Section 13.1 covers the area of sign manipulations while Section 13.7 handles the use of the GKA signal and the sign evaluations necessitated by this early start. To avoid redundancy, only a cursory discussion of these factors is included at this time. Recall that GKA will be started at this point only if the sign of the address of the final operand is positive. This sign is reflected both by the WS and SM flip-flops (WS low for positive, SM high). Before QS is started, the SM flip-flop must contain the sign of the operation: SM high indicates the summing of two positive or two negative values, SM low a differencing. For the single operand commands, the state of SM remains high unless the operand accessed is negative (B28 high) during the KCL loop. This effects the correct setting of SM since the positive opcodes

(A0, T0) would call for a differencing operation if the operand were negative, a summing otherwise. For the odd, or negative, opcodes, SM is reset in the KZC loop. Then, if the operand accessed is negative, the reversal of the state of SM will correctly call for a summing operation. (The sign in WS will be made negative during the operation of QS.) For the two operand commands, the path AS(0)SM is enabled for the odd-numbered (negative) commands, the path AS(C)SM for the positive commands. The reason for the complementation in the latter case is that AS is high to imply negative, while SM is high for positive. Thus, a reversal of states between AS and SM keeps the sign the same which is correct for positive commands. On the other hand, the copying of states from AS to SM reverses the sign value, which is correct for the negative opcodes. In all cases, accessing of a negative operand during the KCL loop will cause the SM flip-flop to be reversed. (Refer to Chapter 13 for a detailed explanation of these manipulations.)

If the address is negative or the QM sequencer is not ready:

(1) the SM flip-flop is set in case the address turns out to be a negative zero; (the sign manipulations at KZH are identical to those at KZD and, thus, assume SM set.) If the address is not zero, it is illegal and the state of SM will be ignored.

(2) a check is made for negative zero by means of the zero test associated with register S. (The address was transferred into S at KZC by means of the N(L3)S path.) If this is a zero address, the sign is complemented and processing continues as stated above since positive or negative zero is a legal address.

(3) if QM is not ready, KC idles between KZG and KZH waiting for it.

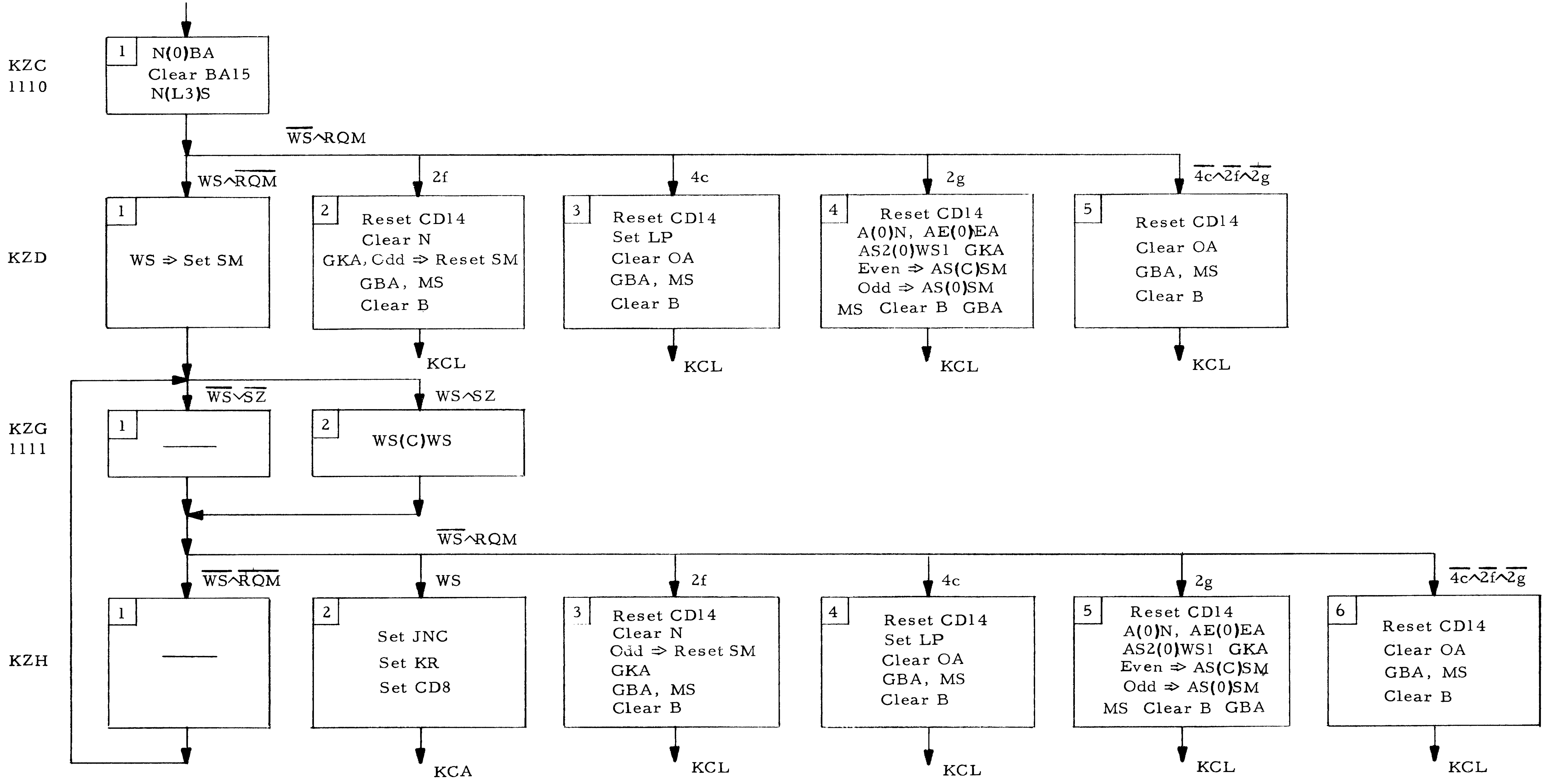
(4) if the address is truly negative, the TILT exit is taken: an interrupt is requested and KC is cleared.

TABLE 9.6–1 Terms Used on the Flow Chart of the KZC Section of Master Control

AS	Accumulator Sign flip–flop, AS is low to indicate positive sign; AS holds the sign of the operand stored,in the Accumulator.
AS(0)WS	Path enabled to transfer the sign of the operand held in the Accumulator to WS; this is necessary when an early start signal is sent to the KA sequencer (GKA). See Section 13.1 and 13.7.
CD14	Bit 14 in register CD (Command Decoding); the state of CD14 is determined by the 29th bit of the command word and, thus, contains made information after command access; if CD14 is high, the mode is 2 or 3, and the KZC loop is entered; thus, CD14 must be reset before opcode startup in order to avoid re–entry into KZC.
Clear B	Register B is cleared prior to startup of the QM sequencer to access the final operand because the transfer into B is single–sided.
Clear BA15	Bit 15 in register BA is cleared simultaneously with the transfer of the 15–bit address (the assembled operand) to 14BA0 to insure a zero in that bit position.
Clear OA	Clear N, EA, reset WS, set SM.
Fix SM	This refers to the determination of the correct state of the SM flip–flop for the case where the KA sequencer is started early. Refer to Sections 13.1 and 13.7 for explanations of sign value determination and the GKA signal.
GBA	START signal to the QM sequencer when the address is in register BA.
GKA	Early GO signal to the KA sequencer which allows for use of the QS startup at KCM to perform the sum or difference called for by the opcode. This is possible for mode 2 or 3 commands since, on the final access, the startup of QS for operand assembly would be meaningless. See Sections 13.1 and 13.7 for a detailed explanation of the GKA signal.
Group 2f	Term used to gate the following commands through Master Control: arithmetic operations A0, A1 (CLA, CLS); arithmetic tests T0, T1 (FOP, FOM).
Group 2g	Term used to gate the following commands through Master Control: arithmetic operations A2–7 (ADD, SUB, ADN, SUN, ADA, SUA); arithmetic tests T2–7 (FSP, FGO, FSM, FLO, FSN, FUO); address preparation N2–7 (OAD, OSU, OAN, OSN, OAA, OSA); logic operations L2, L3 (ADL, SUL); logic tests S2, S3 (ISN, IUO).
Group 4c	Term used to gate the following commands through Master Control: logic operations L0, L1, L4–7 (CAL, CCL, EXL, ECL, UNL, UCL); logic tests S0, S1, S4–7 (IOZ, ICZ, IEZ, IEC, IUZ, IUC).
JNC	JNC flip–flop is set to request an interrupt when a negative address is detected.
KR	Opcode DONE signal to KC.
LP	Logical Product flip–flop; LP is set during the operation of Master Control to gate the commands in group 4c for the final access when they are written in mode 2 or 3: when LP is set, the final access will be made in logic format.
MS	Memory Start signal necessary to the initiation of any memory operation.
N(0)BA	Path enabled to transfer the assembled operand to the BA register for use as an address during the access of the final operand.
N(L3)S	Path enabled to send the assembled operand to register S to be tested against zero.
RQM	READY QM sequencer; no memory operation can begin until this signal is high.
SM	SuM flip–flop; SM high directs QS to perform a summing operation; SM low, a differencing operation.
SZ	S register Zero flip–flop; SZ is set when the operand in register S is zero.
WS	Working Sign flip–flop; WS contains the sign of the operand stored in OA; WS is reset to indicate a positive sign.

FIGURE 9.6–1 The KZC Section of Master Control

KC



SECTION 9.7 – ENTRY AT KZE

The flexibility allowed in the handling of index information forces a general discussion of the subject before the specific operations of the KZE section are spelled out. Two types of addressing dictate the handling of index information. In standard addressing, the contents of the designated index are used to modify the operand. The term special addressing refers to those commands which use the index field to specify the index or register which is to be acted upon in accordance with the particular command. (Special addressing is used with the eight index commands and the four register commands.)

KZE handles all commands which require the accessing of an index, whether written with standard or special addressing. Thus, KZE handles any command written with standard addressing which contains index information. It does not, however, handle all commands written with special addressing, since not all of these require accessing of an index location. For the register opcodes, the index portion of the command word is used to specify the register to be acted upon; thus, accessing of an index is not required and KZE is never entered for these commands. Of the eight available index commands, only four require an index access and, thus, handling by KZE. Explanation of this requires an account of the rule followed in processing all index commands.

Index commands can be grouped into several different categories:

(1) four test commands (B4, B5, B6, B7), and four non–test,

(2) four positive commands (B0, B2, B4, B6), and four negative,

(3) four commands which modify the contents of a particular index by the amount of the operand (B2, B3, B6, B7), and four which load the quantity specified by the operand into the location

named in the I field.

The processing of these opcodes proceeds as follows: the (A) of the command word are added to (OA); if the command is written in mode 2 or 3, the final operand is accessed. At this point a branch is taken for the four opcodes in group 3f: B2, B3, B6, and B7. (These are the index opcodes that call for modification of the contents of an index.) Of the opcodes in this group, two are the opposite of the other two. The working sign is complemented in the two negative cases leaving, in effect, two positive opcodes: B2 and B6. The contents of the indicated index are then accessed and added to the operand. The result is the quantity to be loaded into the index. Recall that the other four index opcodes load a specified quantity, held in register N, into a designated index. The operand now assembled for B2 and B6 is also held in register N and is ready to be loaded into the index. From this it can be seen that these commands now call for the same operations as B0 and B4: loading a positive quantity into an index. This makes possible a further simplification in decoding with B2 becoming B0 and B6 becoming B4. Thus, the four opcodes of group 3f (B2, B3, B6, B7) are first reduced to B2 and B6 and then reduced out of existence by means of changes to B0 and B4. In this way, the next path taken at decision state KCT will be that gated by the 3c group: B0, Bl, B4, B5 and KZE will not be entered again. (It should be pointed out that B4 and B5 are the same as B0 and Bl except that B4 and B5 call for testing the operand against zero.)

The signs for two of the negative cases in group 3f have already been adjusted. In the KCS block, wherever 3c is gated for opcode startup, the working sign is complemented for Bl and B5 (the opposites of B0 and B4). (This change does not affect the B3 or B7 negative commands since by this time they are being decoded as B0 or B4 commands; in this way double reversing of the sign is avoided.) With this last change of sign, the effective number of commands was reduced to two positive cases, one of which calls for a test against zero. The KP sequencer is then started. KP performs the test in the desired case (the address in CA is incremented by 1 when the integer is found to be zero) and stores the operand in the designated register.

To return to KZE, this block is entered at the beginning of this sequence of events for all cases requiring accessing of an index. As stated before, this happens (1) when standard addressing is used and the command contains index information (the contents of the index register are used to modify the operand) and (2) when special addressing is used and the contents of a particular index are to be modified (opcode group 3f). These branches are gated to KZE from KZT by means of the following terms:

(1) $\overline{XZ} \wedge \overline{g3}$

This indicates that the index portion of the command word is not zero and the command is not one of the g3 commands. This latter term excludes all commands which use the I field to specify the index or register to be acted upon by the command. Thus, this branch handles only commands written with standard addressing which contain index information.

(2) $\overline{CD14} \wedge 3f$

Four of the commands excluded in (1) above (B2, B3, B6, B7) are gated by the second branch at KCT if the command is mode 0 or 1. (When the mode is 2 or 3 for these commands, the final access for the desired operand must be made at KZC along with the change of decoding to mode 0 or 1 before this branch is taken.)

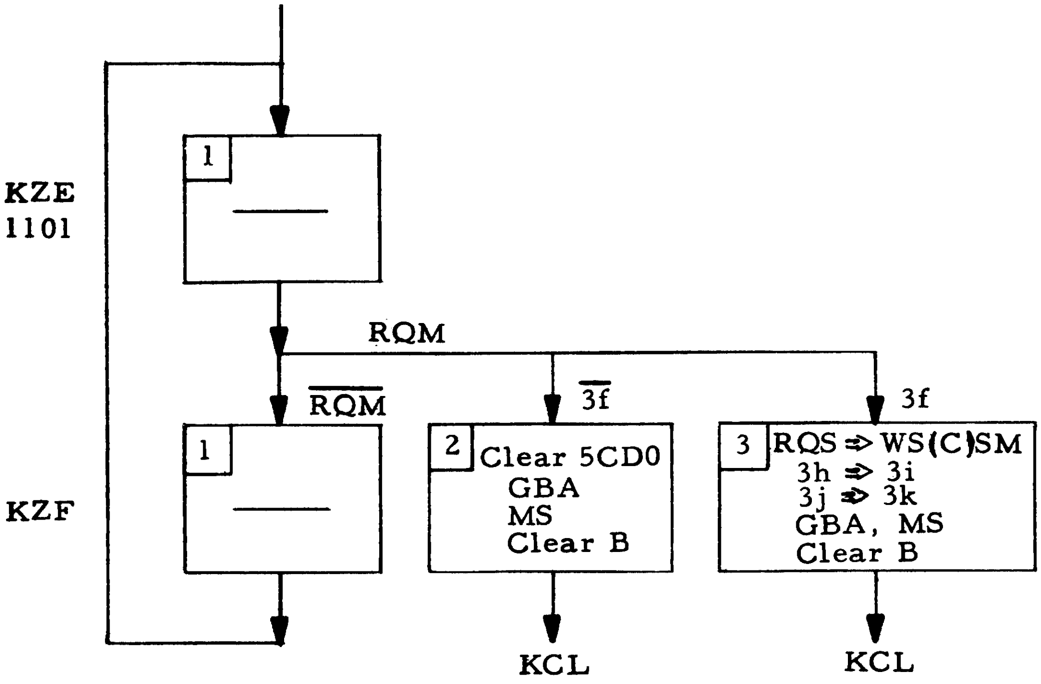
These two branches enter KZE and remain distinct, the gating terms being 3f or $\overline{3f}$ which is sufficiently exclusive to distinguish between the commands which are allowed to enter KZE. The $\overline{3f}$ commands (case 1 above) require clearing of the index portion of the command word and accessing of the index information.

TABLE 9.7–1 Terms Used on the Flow Chart of the KZE Section of Master Control

Clear B	Clear register B; this is necessary before any memory access occurs since the transfer into B is single–sided.
Clear 5CD0	Clear the least significant 6 bits in register CD; these are the bits containing the index address from the command word; once this address has been used during operand assembly, the address must be destroyed so that the same access won't occur again the next time around (XZ will be high).
GBA	START signal to QM sequencer when the address is held in register BA.
Group 3f	Term used to gate the following index commands in Master Control: B2, B3, B6, B7 (ADX, SUX, AXT, SXT).
Group 3h	Term used to indicate the following index commands in Master Control: B2, B3 (ADX, SUX).
Group 3i	Term used to indicate the B0 (LXP) command in Master Control.
Group 3j	Term used to indicate the following index commands in Master Control: B6, B7 (AXT, SXT).
Group 3k	Term used to indicate the B4 (XPT) command in Master Control.
RQS ⇒ WS(C)SM	At READY QS, the sign of the resulting operand, which is held in the WS flip–flop, is copied into the SM flip–flop. This happens in the QS sequencer and is referred to here only to indicate that SM is properly undated. For a discussion of the determination of sign values, see Section 13.1.

FIGURE 9.7–1 The KZE Section of Master Control

KC



SECTION 9.8 – ENTRY AT KCL

This state is entered following each memory access involved in the assembly of the operand. (Accessing of commands is carried out in the KCA section.) Two functions are handled here: the information just accessed is formatted and the QS sequencer is started. Generally, QS adds the new information into the operand. However, if KA has been started early, (see Section 9.6 for further clarification of the early start signal to KA), this GQS initiates the add/subtract stipulated by the opcode, i.e., the operand that resulted from the assembly process is added to or subtracted from the contents of the Accumulator.

The formatting of operands as they are stored in memory is described in Section 11.3. When accessed, the information will be formatted in accordance with the requirements of the opcode and certain information bits in the word itself.

(1) If number format is called for, data flags are enabled ("enable JDA, JDB") so that an interrupt requested through use of a data flag can be processed. If flip–flop B28 is high, indicating a negative sign, the state of the SM flip–flop is reversed (see Section 13.1, in particular, Table 13.1–2);

(2) If this is the right half of a double precision word, (if flip–flop B29 is high), the right half of the word is formatted as follows:

B(0)20D0
B(R21)EP

or, if flip–flop B27 is high indicating a negative exponent value, the path

B(R21C)EP

is enabled. This makes the negative exponent, which was stored in true form, available for arithmetic operations in complement form. Following this the left half of the word is accessed.

(3) When the left half of the double precision word is available in register B, the following transfer takes place:

B(L21)41D21

This leaves the double precision number property position in register D, with exponent in EP.

(4) If single precision format is called for, the format will be pickapoint if the pickapoint enable flip–flop (UPE) and flip–flop B27 are both high. Register PE contains the pickapoint exponent with the sign value in PE6. (PE6 high indicates negative sign.) If UPE is low, the format is floating point and B27 holds the sign of the exponent. Bits 6 and 7 of the exponent registers are set for negative exponents. These bits serve the purpose of storing exponent overflow information (see Section 13.8). They are set for negative exponents since these are used in complement form.

Pickapoint	Floating Point
B(0)26D0	B(0)20D0
$\overline{\text{PE6}}$ ⇒ PE(0)EP	$\overline{\text{B27}}$ ⇒ B(R21)EP
PE6 ⇒ PE(C)EP	B27 ⇒ B(R21C)EP
PE6 ⇒ Set EP6, EP7	B27 ⇒ Set EP6, EP7

(5) If logic format is called for, i.e., if this is the final access for a mode 2 or 3 logic operation, or an operand access during a repeat operation that is handled by KL, logic flags are enabled ("enable JLA, JLB") so that an interrupt requested through use of a logic flag can be processed, and the following transfer takes place:

B(0)31D0
Clear 41D32

In addition, the EP register is cleared since logic format calls for a zero exponent.

(6) If KA6 is high, indicating that the KA sequencer was started in the KZC loop, or by the KM sequencer (final access for a mode 2 or 3 command, or operand access during a repeat operation that is handled by KA (see Section 13.2) KC is essentially finished at this point. KA6 causes branching at the KCM state for formatting. The term

$$KA6 \wedge \overline{LP} \wedge 2a$$

gates the 2a opcodes, the logic opcodes calling for arithmetic operations and logic formatting. The term

$$KA6 \wedge \overline{2a} \wedge \overline{B29} \wedge \overline{LP}$$

gates single precision formatting for all of the arithmetic operations and tests and all of the OA commands except OCA and OCS. The term

$$B29 \wedge \overline{LP}$$

gates double precision formatting for this group.

(7) If LP is high, the operand is accessed in logic format as called for on the final access of a mode 2 or 3 logic command or the access of each operand during a repeat operation handled by KL (see Section 13.2).

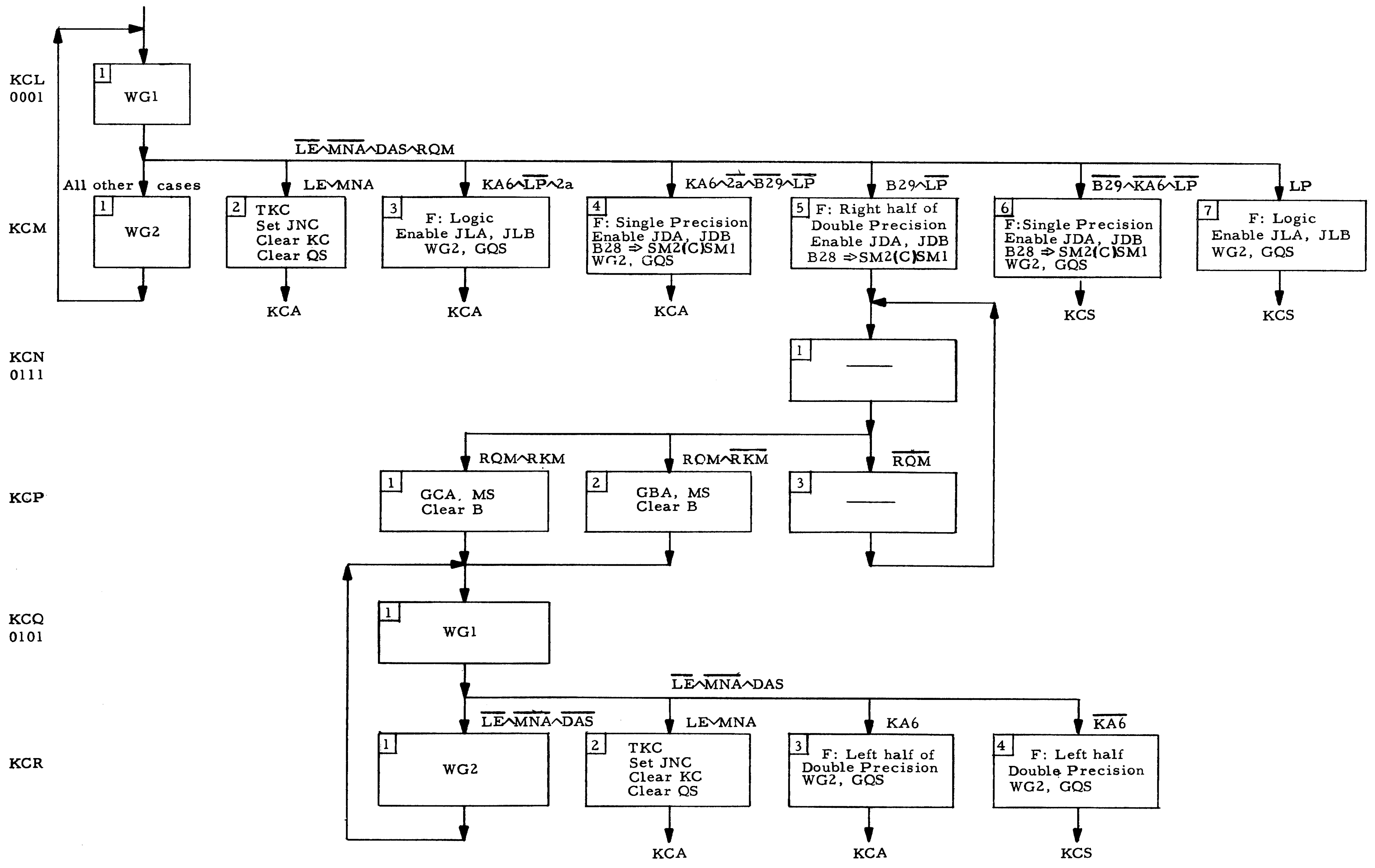
(8) In all cases except the occurrence of errors, completion of this section is accompanied by a startup of the QS sequencer.

TABLE 9.8–1 Terms Used on the Flow Chart of the KCL Section of Master Control

B28	Bit 28 of the B register; for number accesses, this flip–flop contains the sign of the accessed operand; B28 low indicates a positive sign; when B28 is high at KCM, the state of the SM flip–flop is reversed (see Section 13.1 for a discussion of sign manipulations).
B29	Bit 29 of register B; if the operand is accessed in number format, this flip–flop will be set for double precision words, reset for single; for number accesses with B29 high, QM is started to access the second half of the word; if the operand is accessed in logic format, this flip–flop contains the 30th bit of the logic word.
Clear B	The B register is cleared prior to each memory access because the transfer into B from memory is single–sided.
Enable JDA, JDB	Enable detection of data flags.
Enable JLA, JLB	Enable detection of logic flags.
F: Logic, Single Precision, Double Precision	Indicates type of formatting used on accessed information. See text for details.
Group 2a	Term that gates commands L2, L3, S2 and S3 through Master Control.
GCA	GO signal to QM sequencer when the address is held in register CA.
GQS	GO signal to QS sequencer; GQS is used in conjunction with WG2 to start the QS sequencer.
JNC	JNC is set to request an interrupt when MNA or LE is high.
KA6	The KA6 signal is high when KA is in the KAG–KAH idle loop; thus, if KA6 is high at KCM and KCR, KA is known to be in operation. KC bypasses opcode startup and returns immediately to idle.
LE	Large Exponent; this signal is high when QS hangs up due to generation of an exponent overflow condition.
LP	Logical Product flip–flop; LP is set in Master Control to gate final access of mode 2 or 3 logic commands in logic format. This same access path is used to access the operands during repeat logic operations. (See Section 13.2.) LP also affects the operation of the Adder. (See Section 13.1.)
MNA	Memory Non–existent Address; MNA is high when an illegal address is detected during the operation of QM.
MS	Memory Start signal; MS is necessary to the initiation of any memory operation.
RQM	READY QM sequencer; no memory operation can begin until this signal is high.
TKC	Tilt KC sequencer; this signal is high when KC exits from the normal processing paths due to the occurrence of an error condition which results in an interrupt request.
WG1	Wait Gate signal; WG1 is sent to the idle state of QS to enable the N(L3)S path.
WG2	Wait Gate signal; WG2 is used in conjunction with GQS to start the QS sequencer. WG2 is not an echo of WG1.

FIGURE 9.8-1 The KCL Section of Master Control

KC



SECTION 9.9 – ENTRY AT KCS

This is the most complex of the logic sections; it handles basic decision making for the operand assembly and the opcode startup functions. This section follows KCL which starts the QS sequencer prior to its completion. At KCT, the paths taken are determined by answers to the following set of questions:

(1) Is this a legal opcode?

(2) Is index addressing used?

(3) Is it necessary to wait for the completion of QS before continuing with this operation?

(4) Is there any information in the index portion of the command word?

(5) Is this in the mode 0 or 1 or the mode 2 or 3 group?

(6) Has QS finished?

(7) To which group of opcodes does the current one belong? Is the operand properly formatted for this group?

(8) Is this a final access for a mode 2 or 3 logic operation?

These apparently straightforward questions become involved when asked simultaneously, as is the case at KCT. One of the most difficult problems is that of determining which questions have precedence. A further area of confusion results from the repetition of the same questions at KCX and KCZ, since a term not available at KCT may be available at KCX and so forth.

The following signals are used in gating:

At KCS RKM – repeat mode not being used;

At KCT LP – mode 2 or 3 logic commands which use KL; final access is in logic format;

CD14 – modes 2 and 3; $\overline{\text{CD14}}$ – modes 0 and 1;

XZ – index portion of current command contains zeros;

ZE – zero exponent;

DQS – done QS sequencer;

$\overline{WS}$ – positive sign for operand;

It should be noted that ZE can come high only when QS is finished. Thus, the term $ZE \wedge DQS$ would be redundant and only ZE appears in the gating. If, however, QS is finished and the result has a non–zero exponent, both terms $(\overline{ZE} \wedge DQS)$ are needed since $\overline{ZE}$ does not imply DQS.

In the KCS block, the decision states are KCT, KCX, KCZ, and KZK. The requirements of the command word dictate the first decision since special processing is required for accessing of an index register and for commands written in modes 2 and 3. Accessing of indices is handled by the KZE block, modes 2 and 3 by KZC. Assuming that both of these conditions are necessary for a given command, the order of precedence is as follows:

SPECIAL ADDRESSING	STANDARD ADDRESSING
1. KZC	1. KZE
2. KZE (Index commands only)	2. KZC

This is due to the peculiarities of special addressing where, for index commands, the index portion of the command word is used to specify the index that is to be modified by the amount of the operand. In order to accomplish this, the operand must be formed before the value currently in the index is accessed. Thus, the determination of the final operand precedes accessing of the index. For standard addressing, however, the value contained in any specified index location is used to modify the current operand and must be added to it before the final access is made (modes 2 and 3). Thus, the index is accessed before the operand is used as an address to access the final value. If DQS arrives at the KCT state, and the operand is already in integer format, an immediate exit is gated to KZC. Otherwise, two alternatives are available: if DQS has not arrived, the path is gated to an idle loop to await the arrival of DQS; if DQS is high but the operand is not in integer format, the QZ sequencer is started and the path gated to an idle loop to wait for its completion. These gating terms are shown in Table 9.9–2.

When the command word requirements are satisfied, the remainder of the decisions are based upon the requirements of the five basic groups of opcodes. These groupings are as follows:

group 1: the opcodes processed entirely by KC; no K level sequencer is started; the operand is not required to be in integer format and the operand need not be available in order for processing to continue; thus, DQS is not required.

group 2: the opcodes that require an available operand in order for processing to continue; this is because the next step in processing is the startup of K level sequencers and these will use the operand to perform the required operation; thus, DQS is required; integer format is not required for these opcodes. Opcode startup can occur at KCT or KCX.

group 3: the integer and register opcodes (all commands which use index addressing) require the operand in integer format. Opcode startup can occur at KCT, KCX, or KCZ.

group 4: the opcodes which require integer operands (same as requirement for group three, but this applies to standard addressing). Opcode startup can occur at KCT, KCX, or KCZ.

group 5: the opcodes which require that the operand be used as an

address; thus, the operands must be positive integers.
Opcode startup can occur at KCT, KCX, KCZ, or KZK.

The breakdown of each of these groups is given in Table 9.9–1.

Table 9.9–2 lists all of the gating terms at the four decision states (KCT, KCX, KCZ, KZK); on the left side of the chart the terms shown are those required to exit from this block. On the right side are those corresponding to the desired terms which take care of the failure cases. Each alternate path leads, of course, to a reiteration of the required conditions at a later state so that in most cases, the desired terms occur at more than one state. Thus, for all cases where DQS is required and is not available at KCT, gating is provided to the idle loop between KCW and KCX which waits for DQS. The queries are then repeated at KCX.

Two idle loops allow for timing consideration between KC and QS and QZ. If DQS is required and QS is not finished, the idle loop mentioned above is entered.

If QZ is needed to shift the operand, a second idle loop is provided between KCY and KCZ. This idle loop allows cases 3, 4, 5 and 8 to exit immediately on receipt of ZE, the signal indicating that the exponent is now zero. In the case of group 5 commands (which require a positive operand) the presence of a negative sign inhibits exiting from the loop on receipt of ZE. Rather, it calls for a return to KCY and branching on QZA (QZA will be high following sending of the ZE signal since QZ will return to its normal idle loop QZA–QZB). Here the operand will be sent to S for the zero test. At KCZ, the terms gating box 5 will now be available (since Ready QZ = QZB) and opcode startup can proceed if the address is zero. If the operand is non–zero, an interrupt is requested.

It should be noted that when opcode startup occurs at KCT or KCX, exponents are adjusted in accordance with the timing of QS. That is, if DQS arrives at KCT, indicating (N) = 0, the appropriate final transfer ES(0)EA is accomplished by KC. Similarly, at KCX, DQS and opcode startup cause KC to transfer the exponent from ES to EP via the path ES(−1)EP. When QZ is used to shift, the contents of EP are cleared to zero prior to the start of QZ, and the exponents are correct when QZ is done so that final shifts of this type are not required at KCZ.

For several opcode startups, an additional shift of exponents is required: AE(0)EA. This happens when the opcode involves the contents of the Accumulator and the K level sequencer expects to find this exponent value in register EA. (See KCT−4, KCT−5, etc.) Use of this transfer is explained in Table 13.6−1.

One last generalization involves opcodes which are identical except that one is the negation of the other. In such cases, the working sign (WS) is adjusted for the negative case, and processing is identical for both. For example, at KCT−3, the two commands OCA and OCS are gated. If the command is OCS, the sign of WS is reversed by means of the enabling path WS2(C)WS1, after which no distinction is made between the two opcodes.

TABLE 9.9–1 Basic Opcode Groupings

	R				B								N								D		
	0	1	2	3	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2
g1: no opcode startup; opcode processed by KC													x	x									
1a														x									
g2: opcode startup waits for final operand (DQS); integer format not required															x	x	x	x	x	x	x	x	x
2a																							
2b																					x	x	x
2c																							
2d																							
2e															x	x	x	x	x	x			
2f																							
2g															x	x	x	x	x	x			
g3: index addressing; integer operand required	x	x	x	x	x	x	x	x	x	x	x	x											
3a	x	x	x	x	x	x			x	x													
3b						x				x													
3c					x	x			x	x													
3d	x	x																					
3e			x	x																			
3f							x	x			x	x											
3g								x				x											
3h							x	x															
3i					x																		
3j											x	x											
3k									x														
g4: standard addressing; opcode requires integer operand																							
4a																							
4b																							
*4c																							
4d																							
4e																							
g5: operand used as address; thus, positive integer required																							
5a																							
5b																							
5c																							
5d																							
5e																							
5f																							
g6 gates g2, g3, g5 to idle loop waiting for DQS															x	x	x	x	x	x	x	x	x
g7 gates g4, g5 to idle loop waiting for QZA																							
	R				B								N								D		

	A								T								L							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
g1																								
1a																								
g2	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x			x	x				
2a																			x	x				
2b																								
2c		x								x														
2d	x	x																						
2e			x	x	x	x	x	x	x	x	x	x	x	x	x	x			x	x				
2f	x	x							x	x														
2g			x	x	x	x	x	x			x	x	x	x	x	x			x	x				
g3																								
3a																								
3b																								
3c																								
3d																								
3e																								
3f																								
3g																								
3h																								
3i																								
3j																								
3k																								
g4																	x	x			x	x	x	x
4a																					x	x	x	x
4b																	x	x						
*4c																	x	x			x	x	x	x
4d																								
4e																								
g5																								
5a																								
5b																								
5c																								
5d																								
5e																								
5f																								
g6	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
g7																	x	x			x	x	x	x
	A								T								L							

	S								P					X						Repeat	Block
	0	1	2	3	4	5	6	7	0	1	2	3	4	0	1	2	3	4	5		
g1																					
1a																					
g2			x	x																	
2a			x	x																	
2b																					
2c																					
2d																					
2e			x	x																	
2f																					
2g			x	x																	
g3																					
3a																					
3b																					
3c																					
3d																					
3e																					
3f																					
3g																					
3h																					
3i																					
3j																					
3k																					
g4	x	x			x	x	x	x								x		x	x		
4a					x	x	x	x													
4b	x	x														x		x	x		
*4c	x	x			x	x	x	x													
4d																x					
4e																		x	x		
g5									x	x	x	x	x	x	x		x			x	x
5a									x	x	x	x	x								
5b														x	x		x			x	x
5c																				x	
5d																					x
5e														x	x						
5f																	x				
g6	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
g7	x	x			x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
	S								P					X						Repeat	Block

*4c gates these commands written in modes 0 and 1;

LP gates these commands written in modes 2 and 3

TABLE 9.9–2 Gating Terms at KCT, KCX, KCZ, KZK

NO. ON FLOW CHART	TERMS NEEDED TO EXIT	TERMS OCCUR AT:	CAUSE EXIT TO:	REQUIRED ACTION:	DQS	Integer	Positive Integer	REASON FOR FAILURE	TERMS NEEDED TO GATE THIS CASE	NO. ON FLOW CHART	BRANCHES FROM	BRANCHES TO*
I. COMMAND REQUIREMENTS												
9	$\overline{CD14} \wedge 3f$	KCT–8	KZE	Accessing of index								
10	$\overline{g3} \wedge \overline{XZ}$	KCT–9										
8	$CD14 \wedge ZE \wedge [g3 \vee (\overline{g3} \wedge XZ)]$	KCT–13 KCX–9 KCZ–5	KZC	Final access – modes 2 and 3	✓	✓		QS not finished	$CD14 \wedge \overline{DQS} \wedge [g3 \vee (\overline{g3} \wedge XZ)]$	11	KCT–11	1
								Non–integer operand	$CD14 \wedge DQS \wedge \overline{ZE} \wedge [g3 \vee (\overline{g3} \wedge XZ)]$	13	KCT–12 KCX–9	2
II. OPCODE STARTUP REQUIREMENTS												
14	LP	KCT–1	KCA	Opcode Startup								
1	$\overline{CD14} \wedge XZ \wedge g1$	KCT–3	KCA	Access next command								
2	$\overline{CD14} \wedge DQS \wedge XZ \wedge g2$	KCT–4 KCX–2	KCA	Opcode Startup	✓			QS not finished	$\overline{CD14} \wedge \overline{DQS} \wedge [3a \vee (XZ \wedge g6)]$	6	KCT–11	1
3	$\overline{CD14} \wedge ZE \wedge 3a$**	KCT–6 KCX–4 KCZ–2	KCA	Opcode Startup	✓	✓		QS not finished	See No. 6	6	KCT–11	1
4	$\overline{CD14} \wedge ZE \wedge XZ \wedge g4$	KCT–7 KCX–5 KCZ–3						Non–integer operand	$\overline{CD14} \wedge DQS \wedge \overline{ZE} \wedge [3a \vee (g7 \wedge XZ)]$	7	KCX–7	2
5	$(\overline{CD14} \wedge ZE \wedge XZ \wedge \overline{WS} \wedge g5) \vee (\overline{WS} \wedge KZK)$	KCT–5 KCX–3 KCZ–1 KZK–2	KCA	Opcode Startup	✓	✓	✓	QS not finished	See No. 6	6	KCT–10	1
								Negative address	$\overline{CD14} \wedge ZE \wedge XZ \wedge WS \wedge g5$	12	KCT–12 KCX–7	KCZ–4
								Non–integer operand	See No. 7	7	KCT–13 KCX–8	2
								Negative address	$\overline{CD14} \wedge ZE \wedge RQZ \wedge WS \wedge g5$	17	KCZ–4	3
15	Illegal Opcode	KCT–2	KCA	Adjust for error								
16	LE (Exp. Overflow)	KCX–1										
18	WS2 (Neg. address)	KZK–1										

* 1 Idle loop for DQS; then to KCX state
2 Idle loop for QZA; then to KCZ state
3 Test for negative zero at KZJ

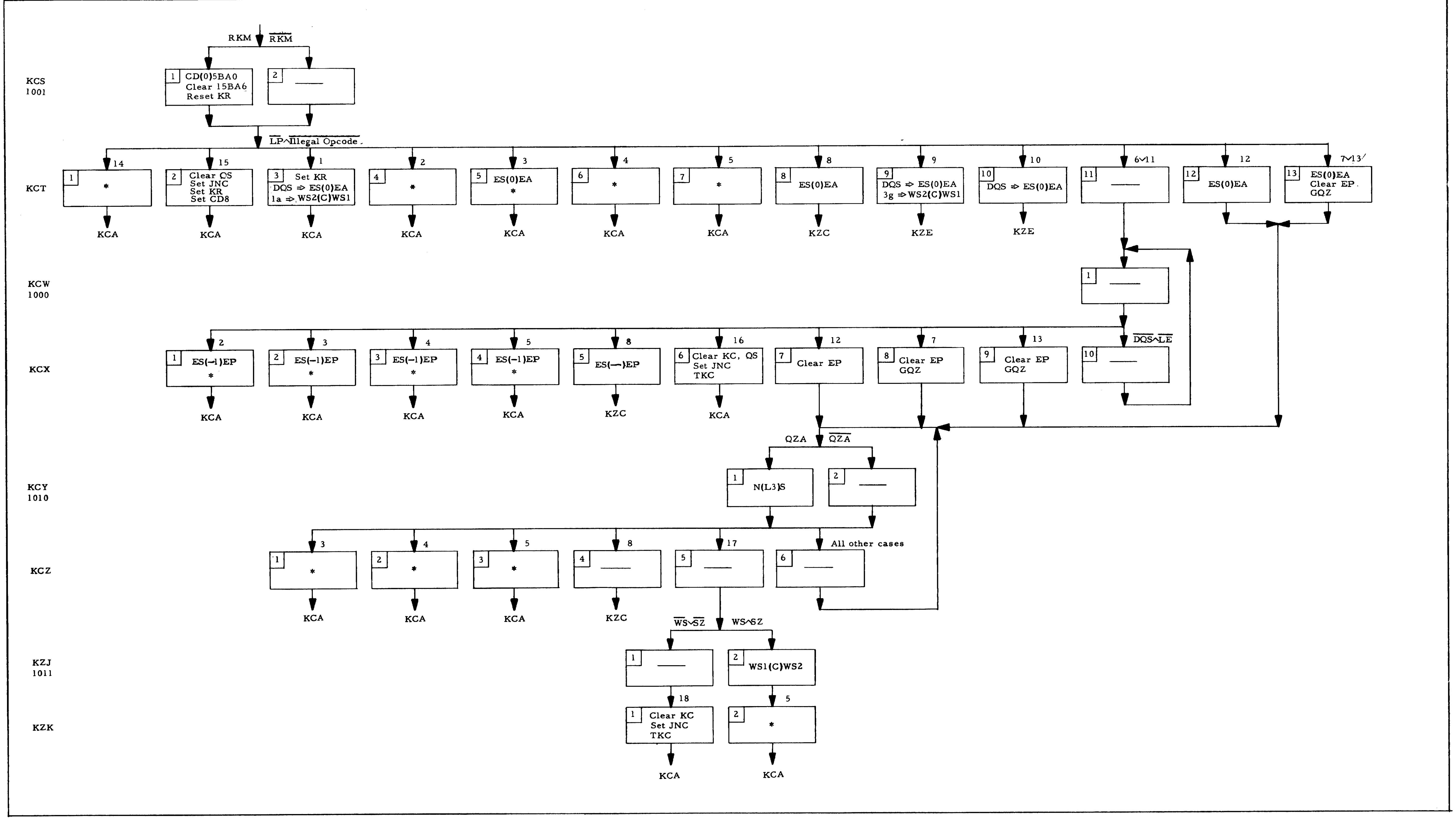
** ZE2 ⇒ DQS

TABLE 9.9–3 Groupings Used on the KCS Flow Chart

GATING TERMS AT DECISION STATES (KCT, KCX, KCZ, KZK)	*OPCODE STARTUP	GROUPS REFERENCED
9 $\overline{\mathrm{CD14}} \wedge \mathrm{3f}$ 10 $\overline{\mathrm{g3}} \wedge \overline{\mathrm{XZ}}$ 8 $\mathrm{CD14} \wedge \mathrm{ZE} \wedge [\mathrm{g3} \vee (\mathrm{XZ} \wedge \overline{\mathrm{g3}})]$ If $\overline{\mathrm{DQS}}$: 11 $\mathrm{CD14} \wedge \overline{\mathrm{DQS}} \wedge [\mathrm{g3} \vee (\mathrm{XZ} \wedge \overline{\mathrm{g3}})]$ If $\overline{\mathrm{ZE}}$: 13 $\mathrm{CD14} \wedge \mathrm{DQS} \wedge \overline{\mathrm{ZE}} \wedge [\mathrm{g3} \vee (\mathrm{XZ} \wedge \overline{\mathrm{g3}})]$ 1 $\overline{\mathrm{CD14}} \wedge \mathrm{XZ} \wedge \mathrm{g1}$ 2 $\overline{\mathrm{CD14}} \wedge \mathrm{DQS} \wedge \mathrm{XZ} \wedge \mathrm{g2}$ If $\overline{\mathrm{DQS}}$: 6 $\overline{\mathrm{CD14}} \wedge \overline{\mathrm{DQS}} \wedge [\mathrm{3a} \vee (\mathrm{XZ} \wedge \mathrm{g6})]$ 3 $\overline{\mathrm{CD14}} \wedge \mathrm{ZE} \wedge \mathrm{3a}$ If $\overline{\mathrm{DQS}}$: 6 If $\overline{\mathrm{ZE}}$: 7 $\overline{\mathrm{CD14}} \wedge \mathrm{DQS} \wedge \overline{\mathrm{ZE}} \wedge [\mathrm{3a} \vee (\mathrm{XZ} \wedge \mathrm{g7})]$ 4 $\overline{\mathrm{CD14}} \wedge \mathrm{ZE} \wedge \mathrm{XZ} \wedge \mathrm{g4}$ If $\overline{\mathrm{DQS}}$: 6 If $\overline{\mathrm{ZE}}$: 7 5 $(\overline{\mathrm{CD14}} \wedge \mathrm{ZE} \wedge \mathrm{XZ} \wedge \overline{\mathrm{WS}} \wedge \mathrm{g5}) \vee (\overline{\mathrm{WS}} \wedge \mathrm{KZK})$ If $\overline{\mathrm{DQS}}$: 6 If $\overline{\mathrm{ZE}}$: 7 If WS: 12 $\overline{\mathrm{CD14}} \wedge \mathrm{ZE} \wedge \mathrm{XZ} \wedge \mathrm{WS} \wedge \mathrm{g5}$ 17 $\overline{\mathrm{CD14}} \wedge \mathrm{ZE} \wedge \mathrm{RQZ} \wedge \mathrm{XZ} \wedge \mathrm{WS} \wedge \mathrm{g5}$ 14 LP 15 Illegal Opcode 16 LE 18 WS	2 { $\overline{\mathrm{2b}}$ ⇒ ES(0)EA (KCT only) 2b ⇒ AE(0)EA, GKD 2c ⇒ WS2(C)WS1 2d ⇒ GQA 2e ⇒ SKA 3 { 3b ⇒ WS2(C)WS1 3c ⇒ SKP 3d ⇒ GKT 3e ⇒ SKJ 4 { 4a ⇒ AE(0)EA 4b ⇒ ES(0)EA (KCT only) 4c ⇒ GKL 4d ⇒ GKJ 4e ⇒ GKX 5 { 5a ⇒ AE(0)EA, GKP 5b ⇒ ES(0)EA (KCT only) 5c ⇒ GKM 5d ⇒ SKM 5e ⇒ GKT 5f ⇒ GKJ 14 { LP ⇒ GKL 4a ⇒ AE(0)EA	g1 = N0, N1 1a = N1 g2 = N2–7, D0–2, A0–7, T0–7, L2–3, S2–3 2b = D0–2 2c = A1, T1 2d = A0, A1 2e = N2–7, A2–7, T0–7, L2–3, S2–3 g3 = R0–3, B0–7 3a = R0–3, B0, B1, B4, B5 3b = B1, B5 3c = B0, B1, B4, B5 3d = R0, R1 3e = R2, R3 3f = B2, B3, B6, B7 3g = B3, B7 g4 = L0, L1, L4–7, S0, S1, S4–7, X2, X4, X5 4a = L4–7, S4–7 4b = L0, L1, S0, S1, X2, X4, X5 4c = L0, L1, L4–7, S0, S1, S4–7, 4d = X2 4e = X4, X5 g5 = P0–4, X0, X1, X3, M0, M1 5a = P0–4 5b = X0, X1, X3, M0, M1 5c = M0 5d = M1 5e = X0, X1 5f = X3 5g = M0, M1 LP ⇒ 4c (modes 2 and 3) g6 = $\mathrm{g2} \vee \mathrm{g3} \vee \mathrm{g5}$ g7 = $\mathrm{g4} \vee \mathrm{g5}$

FIGURE 9.9–1 The KCS Section of Master Control

KC



SECTION 9.10 – REVIEW OF OPCODE STARTUP

Aside from the early start of KA from the KZC block, all opcode startups occur in the KCS block. The six terms that gate these startups are reviewed here.

LP

This term occurs at KCT and gates the same opcodes as the 4c sub–section of group 4 for non–repeat operations. (LP handles commands written in mode 2 or 3, 4c handles mode 0 or 1.) For repeat operations, LP gates each operand access (see Section 13.2). The commands included in this group are:

Logic operations L0, L1, L4–7: CAL, CCL, EXL, ECL, UNL, UCL
Logic tests S0, S1, S4–7: IOZ, ICZ, IEZ, IEC, IUZ, IUC

Opcode startup on the LP branch involves the sending of a start signal to KL, the sequencer which handles all logic operations. No formatting is necessary since, at KCM, the operand gated by LP was accessed in logic format. When the same opcodes are written in mode 0 or 1, they are accessed in number format so that, along with the other opcodes gated by group 4, their startup begins. For the opcodes handled by KL that involve two operands, the exponent of the Accumulator register is sent to register EA via the transfer path AE(0)EA.

$\overline{CD14} \wedge XZ \wedge gl$

These terms gate the OCA and OCS, the commands that are processed by Master Control without startup of a K level sequencer. Since the result may be in any form and no start signal is sent, an immediate exit is made to KCA after KR is set. If QS has finished at KCT, the usual exponent adjustment occurs. It should be noted that accessing of the next command proceeds whether or not the QS sequencer is finished. One additional bookkeeping operation at KCT involves complementing of the working sign if the command is OCS.

$\overline{\mathrm{CD14}} \wedge \mathrm{DQS} \wedge \mathrm{XZ} \wedge \mathrm{g2}$

Group 2 includes the following commands:

Arithmetic operations A0 – 7: CLA, CLS, ADD, SUB, ADN, SUN, ADA, SUA
D0 – 2: MPY, DIV, RDV

Arithmetic tests T0 – 7: FOP, FOM, FSP, FGO, FSM, FLO, FSN, FUO

Address modification N2 – 7: OAD, OSU, OAN, OSN, OAA, OSA

Logic operations L2, L3: ADL, SUL

Logic tests S2, S3: ISN, IUO

These commands have in common the need to have the final operand available before the opcode startup occurs. Integer format is not required. The usual exponent manipulations related to QS take place at KCT and KCX. The MPY, DIV, and RDV opcodes, which are processed by KD, vary from this rule at KCT because the KD sequencer expects to find the Accumulator exponent in register EA. Thus, the transfer path AE(0)EA is enabled at KCT for these three commands. It should be noted that working sign reversal takes place for negative commands.

$\overline{\mathrm{CD14}} \wedge \mathrm{ZE} \wedge \mathrm{3a}$

Group 3a includes the following commands:

register operations R0 – 3: LDR, EXR, ERO, ERA
index operations B0, B1, B4, B5: LXP, LXM, XPT, XMT

This group will gate all of the index and register opcodes since these index commands not specifically included will have their decoding changed (in the KZE block) to allow gating with this group the next time around. At startup, the two remaining negative index commands (B1, B5) cause reversal of the working sign. KP is started for the index commands, KT for LDR and EXR (to transfer information from OA to the specified bus register); KJ is started for register commands ERO and ERA (to perform the extractions indicated). Startup can occur at KCT, KCX, and KCZ depending upon when the operand is available in integer format.

$\underline{\overline{\text{CD14}} \wedge \text{ZE} \wedge \text{XZ} \wedge \text{g4}}$

Group 4 includes the following commands:

logic operations L0, L1, L4 – 7: CAL, CCL, EXL, ECL, UNL, UCL

logic tests S0, S1, S4 – 7: IOZ, ICZ, IEZ, IEC, IUZ, IUC

control operations X2: SKP

input/output operations: X4, X5: TLC, TDC

The logic commands are those gated by LP when written in mode 2 or 3 and by 4c when written in mode 0 or 1. In the latter case, the operand is accessed in numeric format and is shifted to zero exponent before opcode startup. The actual startup is handled in the same way as that of LP: the exponent of the Accumulator is transferred to register EA for those commands involving two operands, and a GKL signal is sent. The KJ sequencer handles the SKP command while KX handles the single character transmit command.

$(\overline{\text{CD14}} \wedge \text{ZE} \wedge \overline{\text{WS2}} \wedge \text{g5}) \vee (\overline{\text{WS}} \wedge \text{KZK})$

Group 5 gates the following commands:

store operations P0 – 5: STD, STS, STL, STI, STZ control operations X0, X1, X3: TRA, TRE, TRM

BLOCK operations M1

REPEAT operations M0

All of these commands require a positive integer operand. Startup may occur at KCT, KCX, KCZ, or KZK. At KZK startup results from testing for negative zero and finding one. When this occurs, the sign is complemented and startup proceeds normally. For the store operations, the exponent of the operand to be stored is sent from register AE to register EA where the KP sequencer expects to find it. The KM sequencer handles block input/output and repeat operations, KT is started for TRA and TRE, and KJ for TRM.

The KA sequencer is started early when the commands processed by that sequencer are written in mode 2 or 3. This startup occurs at KZD or KZH (the latter in the case of a negative zero address). KA is started for the following commands:

2f = arithmetic operations A0, A1: CLA, CLS
arithmetic tests T0, T1: FOP, FOM

2g = arithmetic operations A2 – 7: ADD, SUB, ADN, SUN, ADA, SUA

arithmetic tests T2 – 7: FSP, FGO, FSM, FLO, FSN, FUO
logic operations L2, L3: ADL, SUL
logic tests S2, S3: ISN, IUO
address preparation N2 – 7: OAD, OSU, OAN, OSN, OAA, OSA

KC continues to perform final manipulations after KA is started. In the case of logic operations, KA will leave the result in logic format.

CHAPTER 10

MISCELLANEOUS SEQUENCERS

SECTION 10.1 – INTRODUCTION

Before embarking on a discussion of the remaining sequencers, it would be helpful to relate the tasks performed by the various sequencers to the over–all logical scheme. In Chapter 9, it was pointed out that the three basic tasks performed by Master Control apply to each command word: accessing, operand assembly, opcode startup. During its operation, Master Control pays rudimentary attention to individual opcodes by formatting the operands in accordance with the requirements of groups of opcodes. At opcode startup, the K level sequencers assume the role of controller while KC returns to idle. The sequencer started at this time remains in control until the opcode processing is complete (or nearly so) at which time KC is signalled to begin the next access.

As the processing of a command proceeds, the groupings of those with similar requirements become smaller. Consider the command to add logical and store in the Accumulator (ADL). At access time, it is in the group of all commands that must be accessed. At opcode startup time, it is one of the group of 26 opcodes that require startup of the KA sequencer. During operation of the KA sequencer, it belongs to a group of four opcodes that require startup of QZ in order to shift the result to zero exponent for logic formatting. It might be assumed, then, that minute distinctions between opcodes would be made at the lowest control level, or in the Q sequencers. This, however, is not the case. As a general rule, each Q level sequencer handles a single well defined function which may be used by only one or by several K level sequencers. The Q sequencers whose functions are used by several K level sequencers are generally set up to ignore individual opcodes altogether. When started up, they pick up the required information from specified storage locations and act on it without regard for the source.

This arrangement requires less logic than would be necessary to make it possible for the Q sequencers to differentiate between all cases since, with the K level sequencers handling the distinctions, the breakdown will involve fewer decisions. The best example of this manner of operations is the use of the QS sequencer. When started up, QS performs a summing or differencing operation on the values stored in the N and D, assuming that the respective exponents are in the EA and EP registers, and that the sign of the operation is in SM, and stores the result in OA (N, EA, and sign in WS). QS never takes cognizance of the current opcode; the K level sequencers that use QS set up the operands, exponents and signs before sending the start signal.

All of the K sequencers, with the exception of KT, begin their operation by positioning the operands, starting a Q level sequencer, and idling for the result. Processing often continues with the startup of a second Q sequencer. Consequently, some K sequencers do little more than keep track of the current activities of the Q sequencers, performing bookkeeping operations between their startups. (For example, KD has very little to do with implementation of the multiply/divide algorithms beyond the startup of Q sequencers and the checking of results at various points during the operation. KM does much the same sort of thing during multiple access operations.) On the other hand, KP, while using two Q level sequencers, nonetheless handles a good many of the basic manipulations required for store operations. It appears, then, that the tasks performed by the K sequencers cannot be described as being more basic or of more importance than those of the Q level sequencers whose activities they direct. It is perhaps most reasonable to state that the two distinguishing characteristics of the K sequencers are the factor of control and the orientation toward the needs of the opcodes.

To understand the functioning of the K sequencers, it is necessary to have some grasp of the manner in which the groups of opcodes were formed. With the emphasis on the opcodes, it might be anticipated that opcode groupings for processing would be natural ones, i. e., that each sequencer would handle a group of closely related opcodes. To a considerable extent this is true. However, the explanations for the existence of certain groups under the control of the same sequencer are not always immediately apparent. KT processes two control and two register opcodes, a rather surprising combination, while the index opcodes are handled by the KP, or store in memory, sequencer. Examination of the tasks necessary to the processing of these opcodes reveals the similarities which make them compatible.

The hardware facilities of the G—20 – the Adder, shift paths between registers, subtractor—comparator circuitry, etc. – have been described in earlier sections. The G—20 logic uses these facilities to perform all the functions available to the programmer through the command structure of the computer. The logic is organized into 19 K and Q sequencers. KC (Chapter 9) exerts over—all control; KM, the multiple access sequencer, provides the special control necessary to the processing of block operations; three of the sequencers handle unique operations, KT, QA and QZ. The remaining 14 fall into three basic categories according to the functions they perform and the hardware they use.

1) Those that provide for storing and accessing of words in memory.

2) Those associated with input/output operations which provide communication between the Central Processor and the external world.

3) Those associated with arithmetic and logic operations performed through use of the adder circuitry.

Thus, the discussion is organized as follows:

1) Miscellaneous operations: KT, QA, QZ sequencers. (Because these are both simple and basic they provide a good orientation to the G—20 logic and its documentation.)

2) Memory system: Introduction to the memory system; QM, QB, KP sequencer.

3) Input/output sequencers: Introduction to the input/output system; single character output: KX, Shift—Send mode of QW sequencers; Block input/output: KM, QC, KW, QW sequencers.

4) Adder sequencers: Introduction to opcodes that require use of the Adder; control of repeat operations: KM, QC sequencers; logic operations: KJ, KL sequencers; arithmetic operations: QS, KA, KD, QP, QQ sequencers.

Introductory remarks applicable to all sequencers in each group preceded the sections. In two cases the discussion of a sequencer has been divided between its two functions. KM, the multiple access sequencer, is handled in this manner because its operation in the two instances is quite distinct and its obligations in each case are a part of the functions being described. Similarly, KD's role in divide/multiply operations is most easily understood if discussed in terms of these two functions.

SECTION 10.2 – INTRODUCTION TO KT, QA AND QZ

The three sequencers included in this section are grouped together not because of their similarities, but because of the distinctiveness of the tasks they perform and their consequent lack of conformity to the natural groupings of the other sequencers. The QA and QZ sequencers do have in common responsibility for exponent manipulation: QZ is devoted entirely to the performance of exponent equalization while QA performs the appropriate exponent manipulations prior to the start of the multiply/divide opcodes in addition to its Accumulator Put function. Because the G–20 is a floating–point machine, the handling of exponents takes on great importance. Thus, an early introduction to the QA and QZ sequencers may prove helpful. KT has nothing in common with the other two except that it is also an extremely simple sequencer, comprising, as it does, only two clock times.

SECTION 10.3 – THE KT SEQUENCER

The KT sequencer handles four opcodes, two of which call for a transfer on the basis of the information in OA (the assembled operand), and two of which call for changing the contents of a register on the basis of the information in OA. The latter two are register opcodes: R0 (load register) which calls for the loading of the operand into the selected register, and R1 (extract register) which calls for a selective reset of the indicated register through use of the operand as an extractor. The other two are control opcodes: X0 (GO command) calls for a transfer in the program to the address stipulated by the operand, and X1 (GO and enable interrupts) which is the same as X0, except that it also calls for the turning on of the Master Interrupt Control flip–flop. No diagrams are used in the description of the processing of these commands due to the simplicity of the operations involved and the fact that most of the explanation required involves the significance of the opcodes themselves rather than the means of implementation.

Before opcode startup for X0 and X1 commands, KC checks the assembled operand, which is to be used as an address, for positive sign and zero exponent. If the exponent is non–zero, it starts the QZ sequencer to perform the shifts necessary to make the exponent zero. If the address is non–zero and negative, an interrupt is requested. The register code stipulated for the R0 and R1 commands is checked during the processing of KT (state KTC). It is not checked by KC since the register code is written in the index portion of the command word and is, therefore, sent to the CD register at command access time for future decoding. At KTC, the decoding must be one of those shown in Table 10.3–1, indicating a correct register address; otherwise, an interrupt occurs.

TABLE 10.3–1 Decoding for Register Selection

Signal	Interpretation	Relevant CD Register Bits 11	10	5	4	3	2	1	0
SRU	Select register U	1	X	0	0	0	0	0	1
SRH	Select register H	1	X	0	0	0	0	1	0
SRJ	Select register J	1	X	0	0	0	0	1	1
SPE	Select pickapoint exponent register	1	0	0	0	0	1	0	0

N. B. The X's indicate that this bit is not decoded.

Note that SCA, Select register CA, is not included in the table. This omission results from the fact that this register cannot be loaded using these commands. Its contents are affected by the X0 and X1 commands. Reading of the contents of CA is carried out by means of the R2 (ERO) and R3 (ERA) commands (see Section 13. 4). The 5CD0 decoding for register CA is 000000.

Processing of the two register opcodes depends upon the fact that transfers into register U, J, and H are single–sided while transfers into PE are double–sided. For the command R0, it is desired to load 15 bits of information into registers U, J, and H, and 7 bits into register PE. The quantity to be loaded is the assembled operand. The transfer is accomplished by the enabling of the false and true paths between N (which contains the assembled operand) and the register. For the PE register, only an N(0)PE enable is necessary (double–sided transfer).

The command R1 calls for the extraction of the contents of the register into itself using the assembled operand as the extractor. Recall that the logical function of extracting or ANDing will leave a 1 bit in the result if, and only if, both operands have a 1 in that position. This command applies only to the U, J, and H registers and is used almost exclusively to update the information held in register J, the interrupt request register, when one of the requests has been processed or to reset bits in register U. If the processed request bit were not reset, there would be no way to keep track of work done; on the other hand, the register cannot be cleared because the other requests would be lost while, if time were taken to read out the contents and do the updating in the Arithmetic Unit, a further interrupt request might occur during this process which would be lost. Thus, this command is essential to proper utilization of the interrupt facilities of the G–20. The extraction is accomplished by means of the transfer of the false side of the extractor (N) into the register: i.e., N(F)U. This effects extraction as follows: a 1 bit in the operand will not be transferred; a 1 existing in the corresponding position in the register will, thus, remain. Zeros will be transferred from the operand, thus destroying any 1's in bit positions not containing 1's in the extractor. Thus, a single transfer fulfills the requirements of extraction, leaving the result in the register, and the operation is complete.

The X0 command provides an unconditional jump in the program. This means that the previous contents of the next command register, CA, are to be replaced by the assembled operand and no record is to be kept of the former address. Thus, all that need be done is to enable the path N(0)CA.

X1 performs the same transfer as that of X0 to effect the required jump in the program. KT then resets UJE, the Master Interrupt Control flip–flop, in case it was set already, sends an early start to KC, and waits until KC has advanced to state KCC. This allows time for KC to access the next command and insures that no interrupt will be processed before the next command access. When KCC is reached, KT enables UJE and sends an opcode DONE signal to advance KC.

TABLE 10.3–2 Terms Used on the KT Flow Chart

Command decoding designations:

$CD11$ – Register opcode;

$\overline{CD11}$ – Control opcode;

$CD11 \wedge \overline{CD10}$ – R0 (LDR);

$CD11 \wedge CD10$ – R1 (EXR);

$\overline{CD11} \wedge \overline{CD10}$ – X0 (TRA);

$\overline{CD11} \wedge CD10$ – X1 (TRE);

KC5 KC5 high indicates state KCC of KC sequencer

NRC Non–legal Register Code; NRC is high when the designated register is not one of those shown in Table 10.3–1.

RC Legal Register Code; RC is high when the designated register is one of those shown in Table 10.3–1.

SKC Early START to KC sequencer.

SRH Select Register H.

SRJ Select Register J.

SRU Select Register U.

SPE Select Pickapoint Exponent register.

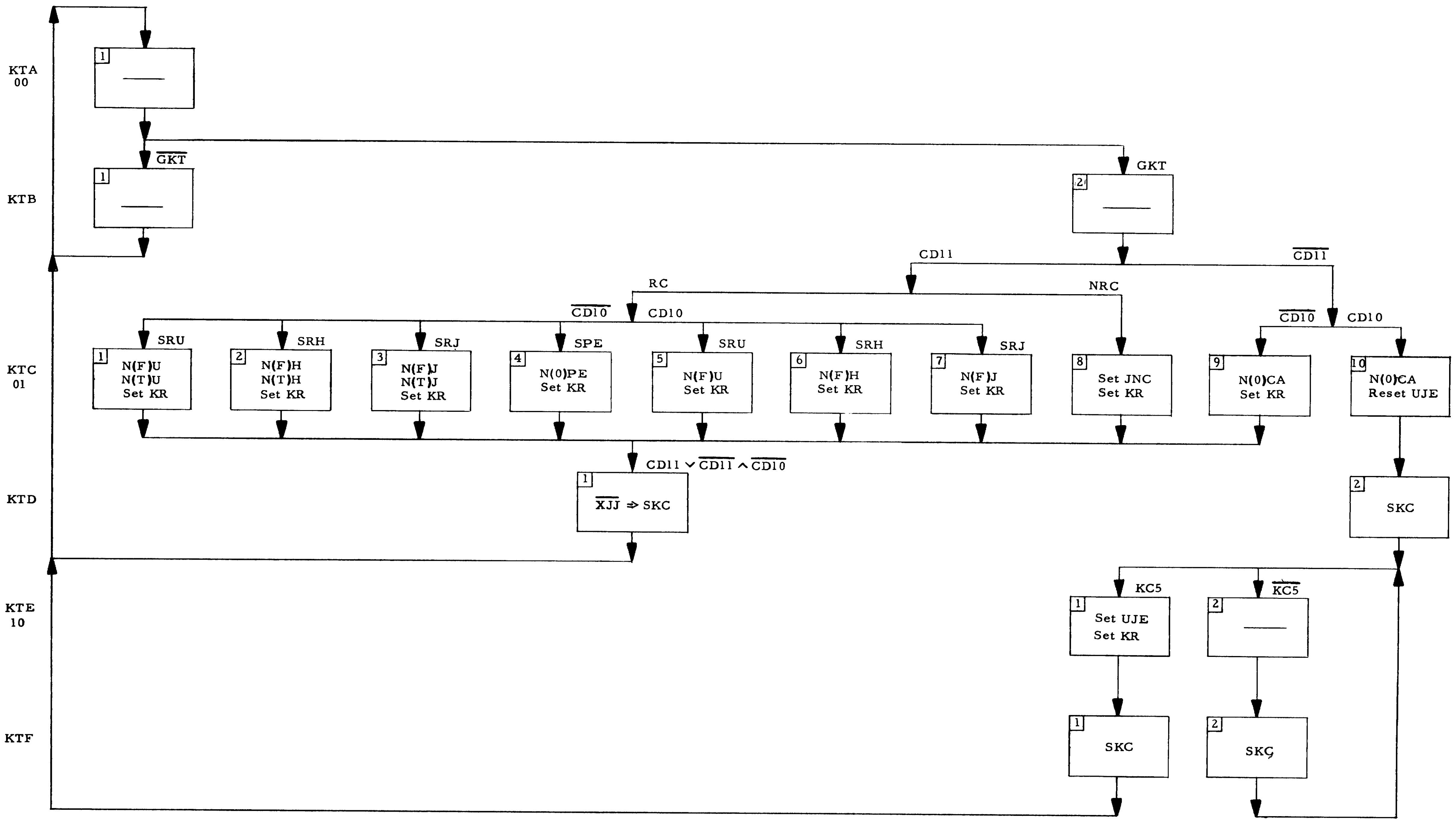
XJJ Enabled interrupt has been requested; SKC is not sent when this has occurred.

Set JNC Request interrupt.

UJE Master interrupt control flip–flop; when UJE is reset, no interrupt requests can be processed.

FIGURE 10.3–1 The KT Flow Chart

KT



SECTION 10.4 – THE QA SEQUENCER

The QA sequencer performs two tasks: accumulator put, wherein it stores the result of an operation from OA to the Accumulator, and exponent manipulation, which calls for specific manipulation of exponents and mantissas for the D opcodes (multiply/divide). The first of these tasks, called the Accumulator Put, is the function for which the sequencer was named. It is used following performance of all opcodes calling for the result to be stored in the Accumulator as well as after the performance of block input/output opcodes which, on termination, leave the address plus one or the address plus two of the last operand in the Accumulator. Since this sequencer will never be started until the processing of the current opcode is complete, QA is charged with a task usually reserved for K sequencers, that of signalling KC to access the next operand through use of the KR signal (opcode DONE).

Implementation of the accumulator put function calls for three transfers:

1) the result left in register N is gated to register A;
2) the exponent of the result is transferred from register EA to the Accumulator Exponent register, AE;
3) the sign of the result is copied from flip–flop WS into flip–flop AS.

The exponent manipulations performed by QA for multiply/divide operations involve the formation of the initial product or quotient exponent through use of the subtractor–comparator circuitry, SC. This circuitry is set up to subtract the contents of register EP from the contents of register EA with the result being gated into ES. The exponents for the three commands in question must be loaded into the EA and EP registers in such a way that this subtraction yields the desired result.

1) Divide calls for the subtraction of the exponent of the denominator from that of the numerator. Since (A) = numerator, (N) = denominator in this operation, the exponents associated with these values are already stored correctly in registers EA and EP. Thus, enabling of SC(0)ES gives the appropriate initial exponent, (EA) – (EP).

2) For reverse divide, the correct term is obtained by complementing the output of SC since $\overline{(EA) - (EP)} =$ $(EP) - (EA)$.

3) Multiplication calls for the addition of the exponent of the multiplicand to that of the multiplier. In order to use the SC circuitry for this purpose, the complement of (EA) is used. Thus,

$$-[(EA)] - (EP)$$

which is the same as

$$-[(EA) + (EP)].$$

Complementation of this result gives the desired result.

4) The exponent in ES is gated into EA; the final exponent sign is stored in EA8 by means of enabling the path

ES7(0)EA8

for divide, and the path

ES(C)EA8

for reverse divide and multiply. The complement path is necessary in these cases since the output from SC will be in complement form.

5) For reverse divide and multiply operations, the contents of A are switched with the contents of N.

6) The sign of the contents of A, held in flip–flop AS, affects the

sign of the contents of N, held in flip-flop WS, if (A) is negative by causing WS to be complemented; this, through enabling of the WS2(C)WS1 path. See Section 13.8 for a discussion of sign determination.

FIGURE 10.4–1 Algorithm for the Accumulator Put Function of QA

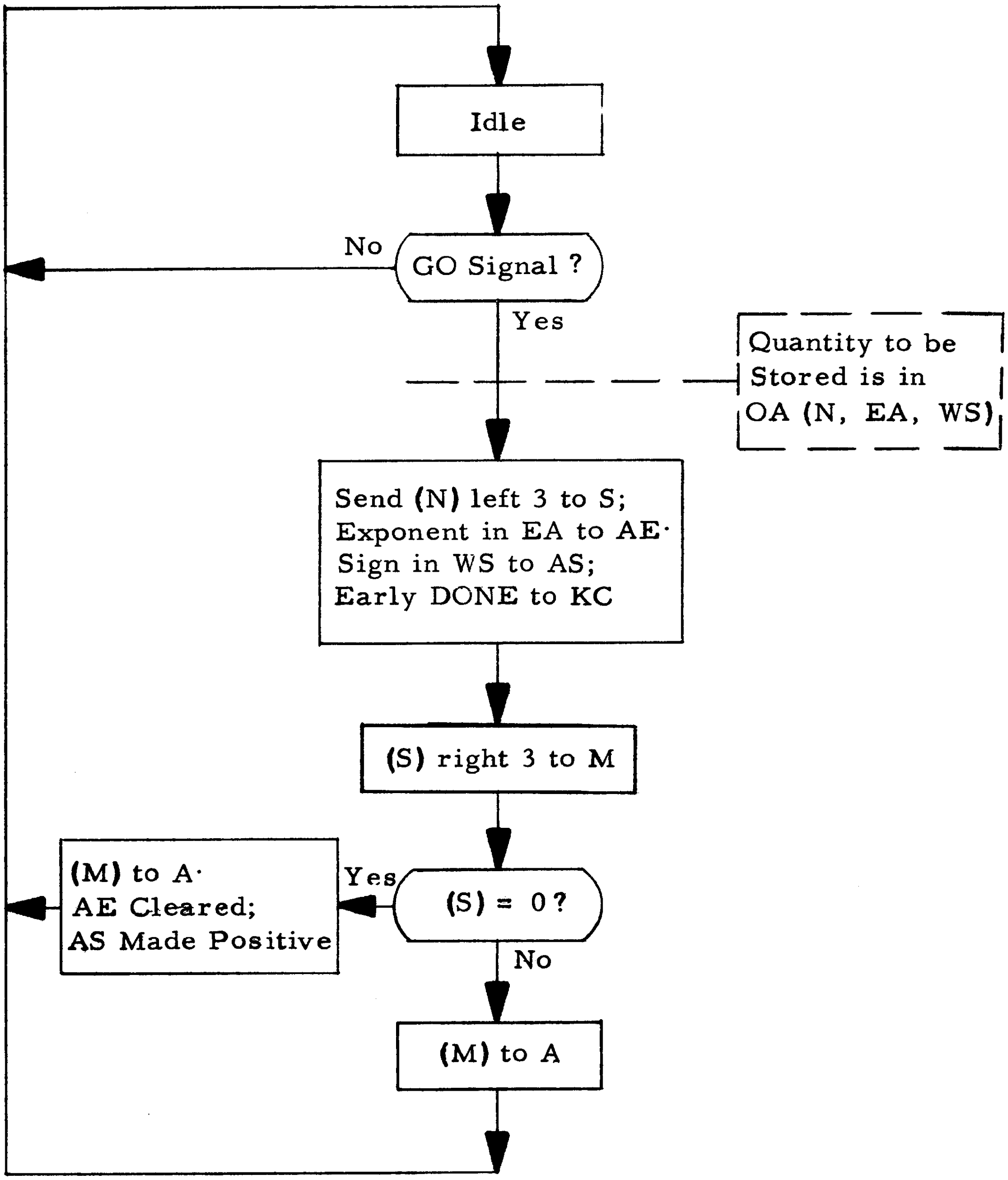

Idle
No
GO Signal ?
Yes
Quantity to be Stored is in OA (N, EA, WS)
Send (N) left 3 to S; Exponent in EA to AE· Sign in WS to AS; Early DONE to KC
(S) right 3 to M
(M) to A· AE Cleared; AS Made Positive
Yes
(S) = 0?
No
(M) to A

FIGURE 10.4–2 Algorithm for QA Manipulations for Multiply and Divide

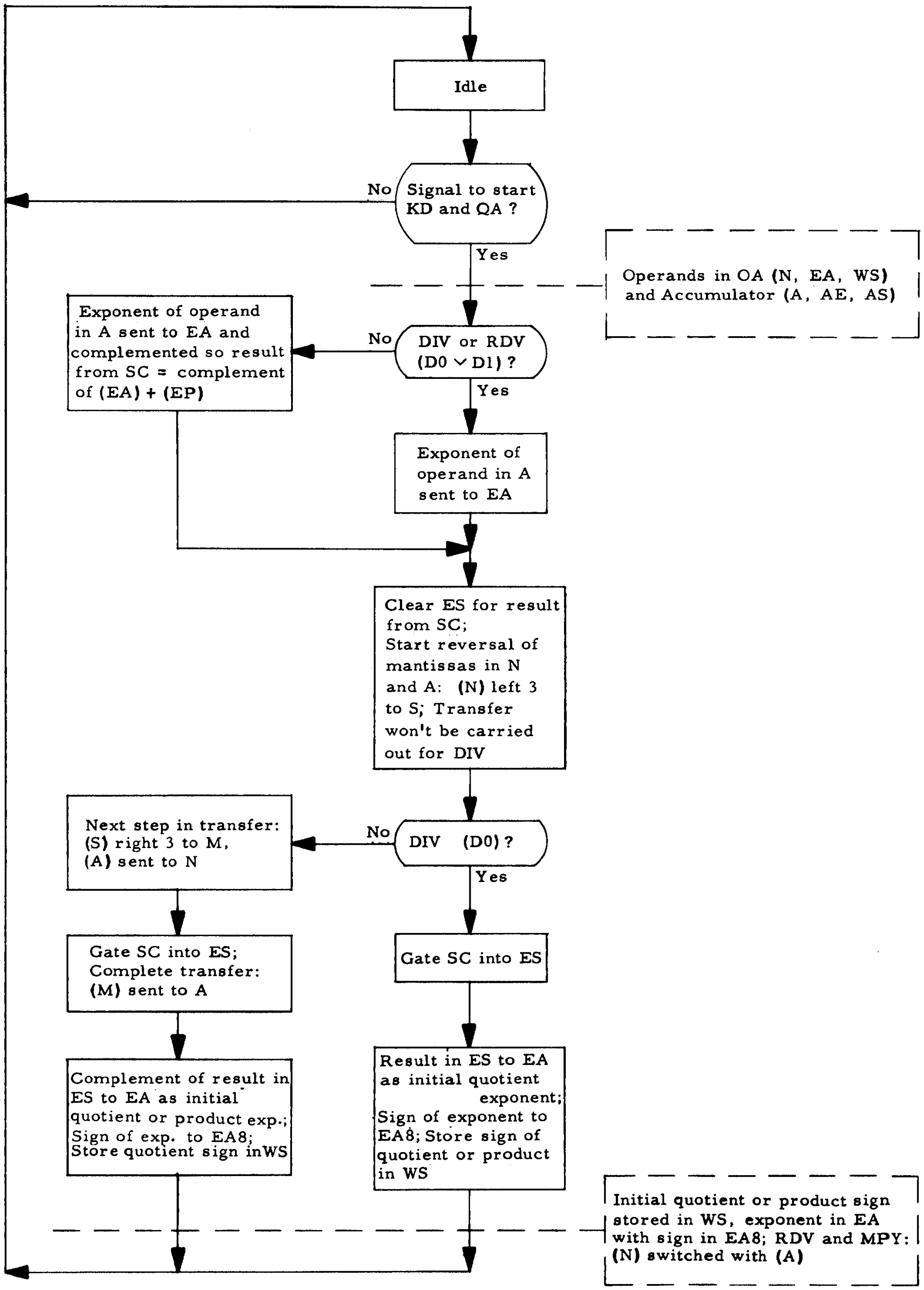
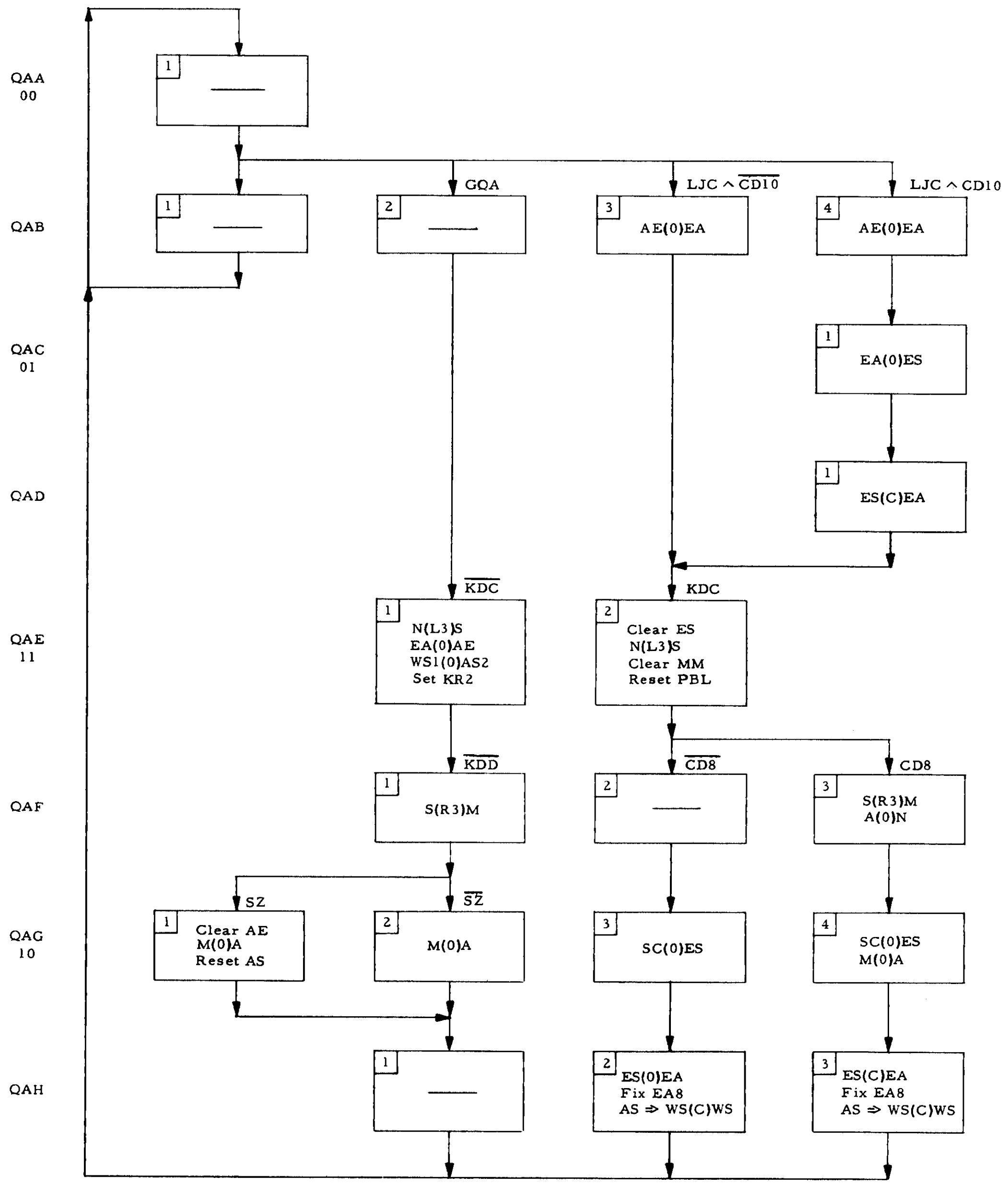

Idle
Signal to start KD and QA ?
No
Yes
Operands in OA (N, EA, WS) and Accumulator (A, AE, AS)
DIV or RDV (D0 ∨ D1) ?
No
Yes
Exponent of operand in A sent to EA and complemented so result from SC = complement of (EA) + (EP)
Exponent of operand in A sent to EA
Clear ES for result from SC; Start reversal of mantissas in N and A: (N) left 3 to S; Transfer won't be carried out for DIV
DIV (D0) ?
No
Yes
Next step in transfer: (S) right 3 to M, (A) sent to N
Gate SC into ES; Complete transfer: (M) sent to A
Complement of result in ES to EA as initial quotient or product exp.; Sign of exp. to EA8; Store quotient sign inWS
Gate SC into ES
Result in ES to EA as initial quotient exponent; Sign of exponent to EA8; Store sign of quotient or product in WS
Initial quotient or product sign stored in WS, exponent in EA with sign in EA8; RDV and MPY: (N) switched with (A)

TABLE 10.4–1 Terms Used on the QA Flow Chart

AS	Accumulator Sign flip–flop; AS high indicates negative.
CD8	Command decoding bit used to distinguish between divide and reverse divide or multiply; CD8 is high for reverse divide and multiply, low for divide.
CD10	Command decoding bit used to distinguish between multiply and divide; CD10 is high for multiply, low for divide or reverse divide.
Clear MM	Bookkeeping operation necessary for multiply/divide; See Section 13.8 for a discussion of the Modulo–Three counter.
Fix EA8	Set EA8 if exponent in EA is negative, reset if positive.
GQA	GO QA sequencer.
KDC, KDD	States of the KD sequencer (multiply/divide).
KR	Opcode DONE signal to KC.
LJC	START QA sequencer signal from KC for multiply/divide opcodes; LJC also starts the KD sequencer.
PBL	Product Bits Lost flip–flop used in processing multiply/divide opcodes; PBL is reset by QA as part of the operations performed for these opcodes.
SZ	S register Zero flip–flop; SZ is high when register S contains zero.
WS	Working Sign flip–flop; WS holds the sign of the operand in OA; WS high indicates negative.

FIGURE 10.4–3 The QA Flow Chart

QA



SECTION 10.5 – THE QZ SEQUENCER

The QZ sequencer performs the function of shifting the current operand (held in N) until its exponent agrees with that held in the EP register. It is started for two purposes: The usual case is that in which it is desired to shift (N) to zero exponent. To accomplish this, the EP register is cleared prior to the startup of QZ so that equalization will be achieved when (N) have been shifted to zero exponent. In the other instance, the command P1 (STS) is used with the pickapoint mode enabled so that the exponent of the value to be stored must be equal to the value currently stored in the pickapoint exponent register. When this happens, the pickapoint exponent is sent to register EP prior to the startup of QZ so that the operand will end up with an exponent equal to the pickapoint exponent. Nine octal digits of the operand will then be stored without an exponent appended, but with a flag in bit 27 to indicate pickapoint exponent, as distinguished from pickapoint integer.

The handling of integers written in the pickapoint mode requires additional explanation. Integer format normally calls for shifting to zero exponent and truncating to seven octal digits. However, it is possible to retain nine octal digits after the shift, if in the pickapoint mode. Thus, pickapoint integers are shifted to zero exponent but retain nine octal digits. This formatting is handled by the KP sequencer which performs final storing for index commands and which formats the operand for the P3 (STI) command (store in integer format). KP does not perform the shift to zero exponent for the index commands; that part of the formatting is handled by KC. KP does use QZ to shift the operand in the case of STI commands, however. In both cases, KP stores seven octals for non–pickapoint, nine octals for pickapoint mode.

There are two other store commands which cause KP to start QZ: STS and STL. The STL (STore Logical) command calls for shifting of the operand to zero exponent; lost bits at either end of the word are disregarded. KP completes the formatting by truncating the operand to 32 bits. Shifting for the STS (STore Single) command is handled by QZ only if the pickapoint flip–flop is on. When QZ is started, in this instance, loss of significant bits due to left shifts is considered an error and results in an interrupt. If bits are lost due to right shifts, the loss is recorded in the discarded bits flip–flop so that KP can roundup before storing the operand.

KC uses QZ to shift the assembled operand to zero exponent in three cases: 1) the assembled operand is to be used as an address, 2) the current opcode requires an integer operand, and 3) the current opcode is to be handled by KL which expects to receive an operand with zero exponent so that it is partially in logic format. (KL performs the truncation to 32 bits that completes formatting.) KC clears EP before sending GQZ and thus effects the shift to zero exponent.

KL, the logic sequencer, starts QZ to shift the operand stored in the Accumulator to zero exponent for the two–operand logic commands (L4–7) and logic test commands (S4–7). The operand that is assembled prior to startup of KL will have been shifted to zero exponent by QZ under the jurisdiction of KC as a part of the operand assembly process. Truncation of both operands to 32–bit logic format is handled by KL following the finish of QZ.

KA, the arithmetic sequencer, uses QZ in the processing of the four commands it handles that require the result to be in logic format. These opcodes (L2, L3, S2, S3) are processed with the arithmetic commands until the result is obtained. The result is then shifted to zero exponent (by QZ) and then truncated to 32 bits to provide the logic formatting.

At GO QZ, the operand to be shifted is stored in register N and its exponent in register EA. If the number is to be shifted to zero exponent, register EP will have been cleared. Otherwise, the pickapoint exponent will have been transferred into EP. Exponent circuitry will then provide signal EE (Exponents Equal) if (EP) = (EA), EL if (EA) $\geq$ (EP), if $\overline{\text{EL}}$ if (EA) < (EP). If EL is high, left shifts are required with concurrent exponent decrements. $\overline{\text{EL}}$ has the opposite affect. When the shift is to zero exponent, the state of EL also indicates whether the current exponent is positive or negative.

The shift paths used to carry out these tasks are listed below. Much more extensive use is made of these same signals and shift paths in the QS sequencer which will be discussed in Section 13.6.

EL: Paths enabled for left shifts: N(L3)S, S(0)N
Paths enabled for exponent decrements: EA(0)ES, ES(−1)EA

$\overline{\text{EL}}$: This calls for right shifts of (N). Since no paths exist between S and N which allow for right shifts, it is necessary to switch (N) with (D) in order to use the right shift paths between D and S. Paths enabled to perform this switch:

N(L3)S, D(0)N, S(R3)D

Paths enabled for right shifts: D(0)S, S(R3)D
Paths enabled for exponent increments: EA(0)ES, ES(+1)EA

EE: When EE goes high, the mantissas must be restored to their original positions. Paths enabled to perform this restoration:

N(L3)S, D(0)N, S(R3)D

The two algorithms that follow show the action taken in the two possible cases: where the operand is to be shifted to zero exponent (Figure 10.5−1) and where the exponent is to be made equal to the pickapoint exponent (Figure 10.5−2).

FIGURE 10.5-1 Algorithm for Shifting the Operand to Zero Exponent

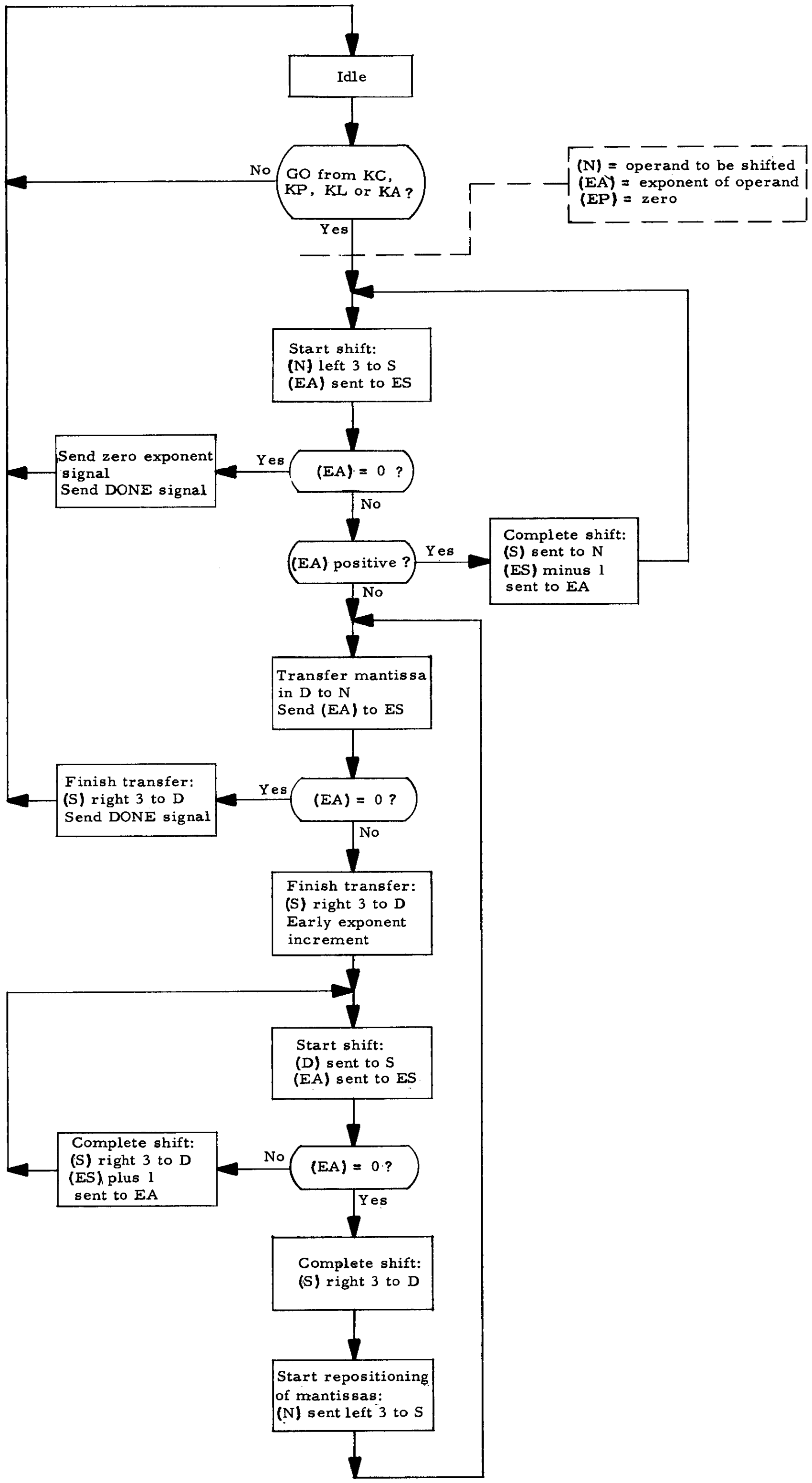


FIGURE 10.5–2 Algorithm for Setting Exponent Equal to Pickapoint Exponent for P1 (STS) Command Pickapoint Mode

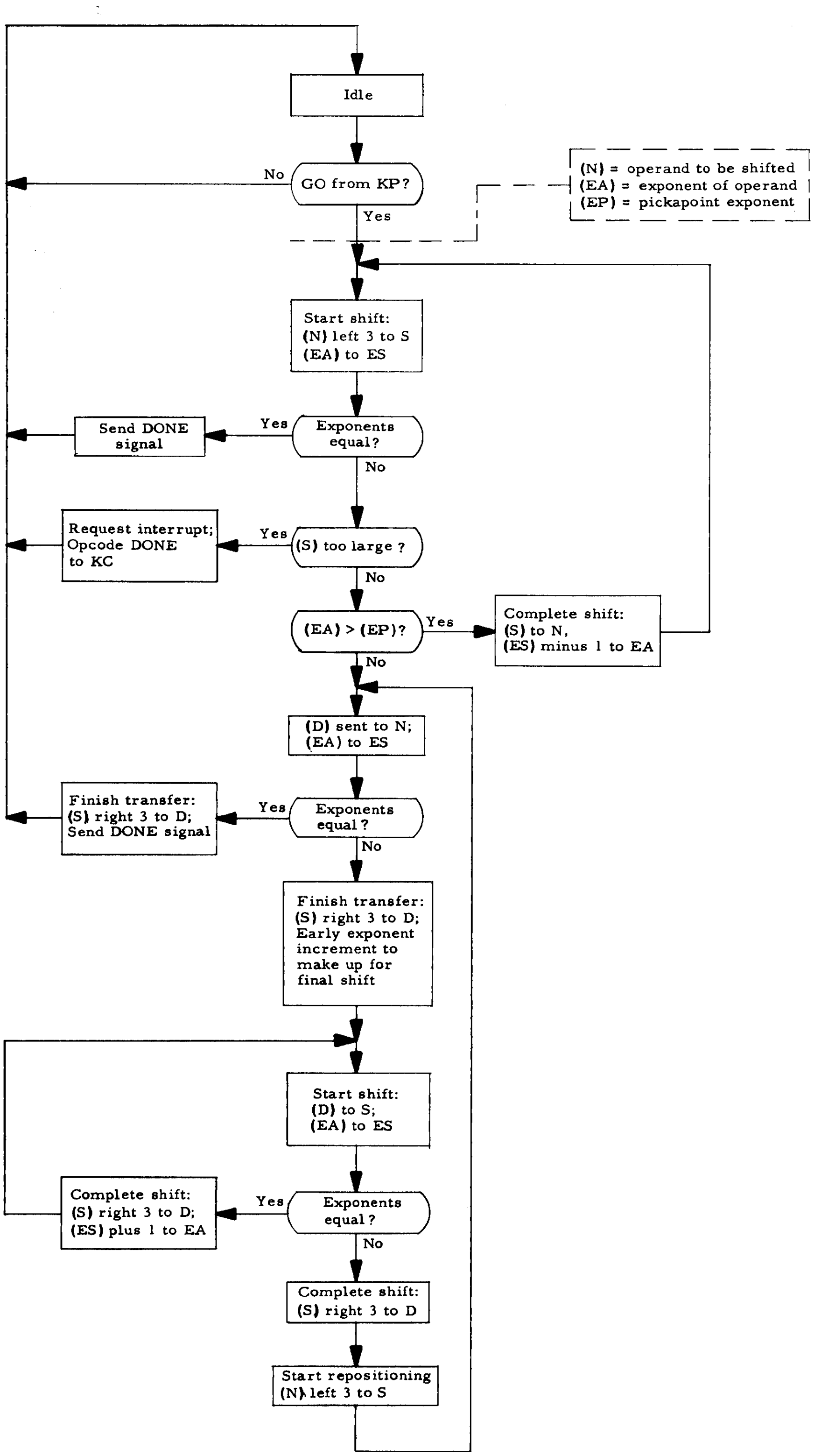
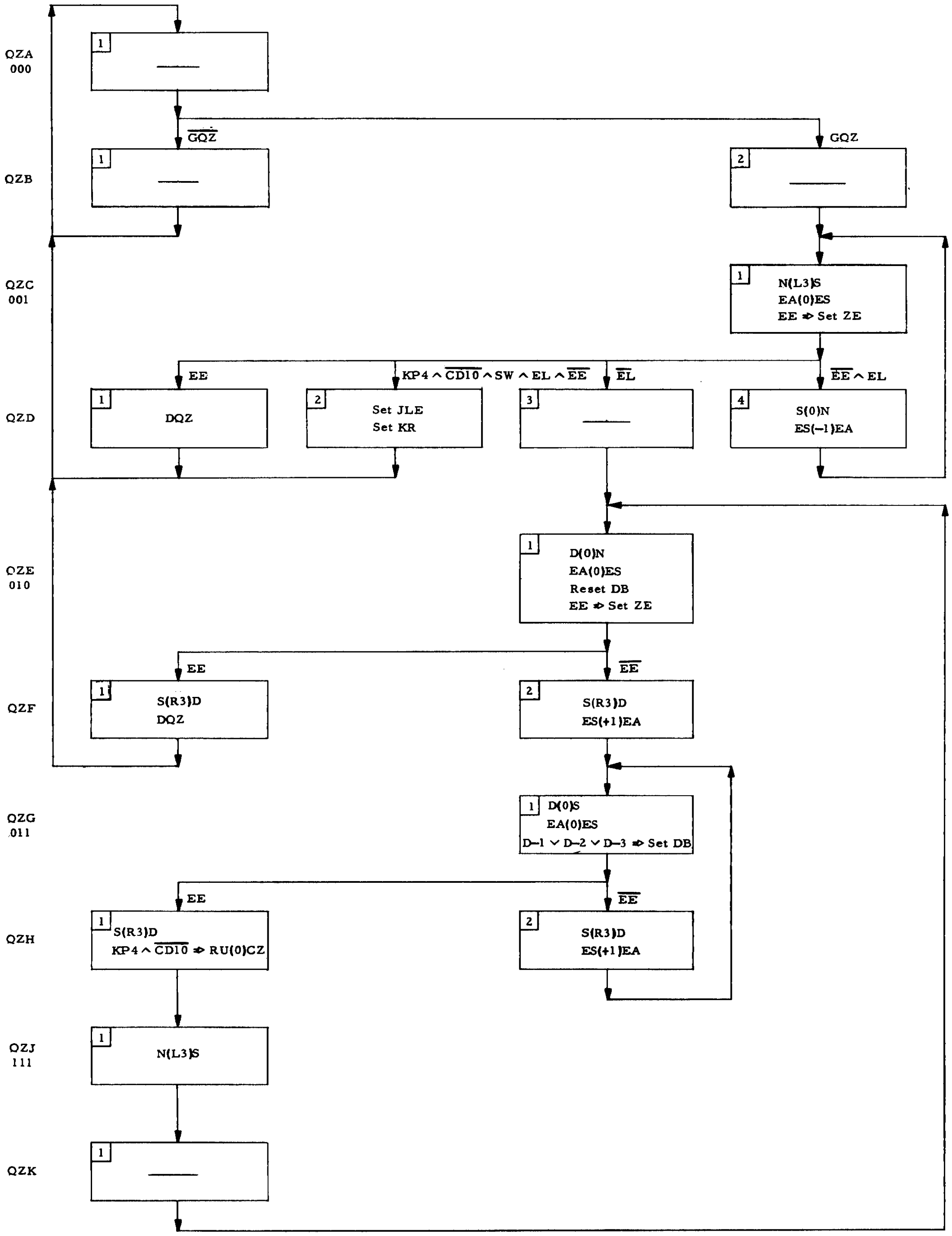


TABLE 10.5–1 Terms Used on the QZ Flow Chart

CZ	Carry Zero flip–flop; CZ is set to cause a carry in to the first bit of the Adder. In QZ, if the roundup signal (RU) is high, indicating that rounding should be carried out by the KP sequencer for the STS command, the CZ flip–flop will be set by enabling of the RU(0)CZ path.
DB	Discarded Bits flip–flop; DB is set when any of the three bits shifted out of the D register on a right shift is a 1; this information is used in establishing the condition of the roundup signal: $DB \Rightarrow D{-}1 \vee D{-}2 \vee D{-}3$
DQZ	DONE QZ sequencer.
EE	Exponents Equal; (EA) = (EP).
EL	Exponent Larger; (EA) > (EP).
$KP4 \wedge \overline{CD10}$	Terms used to gate the P1 (STS) command through QZ in the pickapoint mode.
KR	Opcode DONE to KC.
RU	Round–Up signal; RU is high when rounding is required. The non–biased round–up rule states that round–up takes place if the least significant octal digit (the next one to be discarded by an S(R3)D transfer) is greater than half. If this digit is exactly one–half and the least significant digit retained is even, round–up. Otherwise, truncate. S2 defines one–half since, after the S(R3)D transfer, this will be 2^{-1}. $S0 \vee S1 \vee DB$ indicates greater than one–half if S2 is high, and S3 indicates that the least significant digit retained is even. Thus, $RU \Rightarrow S2 \wedge (S0 \vee S1 \vee \overline{S3} \vee DB)$ If RU is high, CZ will be set to effect the round–up.
Set JNC	Request interrupt.
SW	S register Wide signal; SW is high when the operand in register S has overflowed, i.e., there are significant bits in $S44 \vee S43 \vee S42$.
ZE	Zero exponent flip–flop; ZE is set when the EA register contains a zero exponent and QS is done.

FIGURE 10. 5–3 The QZ Flow Chart

QZ



CHAPTER 11

MEMORY SEQUENCERS

SECTION 11.1 – INTRODUCTION TO MEMORY CONTROL LOGIC

The G–20 memory system has been designed to provide efficient operation given any of the possible system configurations. G–20 memory modules consist of panels of 4,096 words each; a single computing system can contain no more than eight such panels. The portion of the basic G–20 computing system that is referred to as the external memory system consists of from one to seven memory panels and an associated bus system. The memory system panels are housed in MM–10 units, one or two in each. This part of the memory is called external memory to distinguish it from the one or two panels internal to the system Central Processor. Internal memory is not a part of the memory system; that is, it is not connected to the bus system and it cannot be accessed by any unit other than the Central Processor. The bus system connecting the external memory units to the memory users is comprised of four buses: one for the transmission of control signals and three information buses (an address bus for the transmission of necessary address information, a write bus to carry the word that is to be stored in WRITE operations, and a read bus to send out the word that has been read on READ operations). Thus, each memory operation involves the use of three buses.

In the basic G–20 system there are one, two, or three memory users:

a Central Processor and, at the option of the user, one or two two-channel DC–11 modules. Each of these units contains the logic necessary to control memory operations and is capable of initiating these operations independently. In an expanded G–20 System, all memory users (additional G–20's and DC–11's) will contain the logic necessary to effect such control.

External memory units have been designed to function independently of one another in order to make possible simultaneous use of two or three MM–10's. The limiting factor on the number of simultaneous memory operations possible within a memory system is the time involved in the use of the bus system. A bus is tied up for two microseconds on each use; thus, if three controllers required access to three different MM–10's, startups could occur at two microsecond intervals, as the address bus became free. Any time two controllers required use of the same MM–10, the second one is, of course, held up until the first has completely finished the memory operation. (An MM–10 can perform only one memory operation at a time.) A priority system determines which user gets precedence if two or more of them request use of the bus simultaneously.

A memory operation, once started, is controlled by the addressed memory timing counter, that is, the timing counter associated with the panel containing the addressed location. Each timing counter carries to completion the memory operation initiated in the associated panel. Memory start signals to G–20's or MM–10's containing two memory panels start both timing counters, but only the counter of the addressed panel remains in operation. Timing counter control of memory operations is an important factor in the independence of MM–10 modules. Equally important are the 15–bit MA register and the 33–bit B register contained in the Central Processor and the 16–bit MA and 33–bit MB registers of the same description in the DC–11.

(The DC–11 is provided with a 16–bit address register in anticipation of an expanded G–20 system in which 14 memory panels would be available in external memory. Addressing of this number of words requires 16 bits.) When a memory user requires a memory operation, it stores the address of the designated location in its internal MA register.

From its MA register, the Central Processor decodes the three most significant bits. These bits are sufficient for the determination of the addressed memory panel since the panel containing the lowest addresses (0 – 7777) will have 000 in 14MA12, the second panel, containing 10,000 – 17,777 will have 001, etc. The remaining 12 bits designate the address within the indicated panel. Thus, when decoding of 14MA12 indicates that an external memory operation is called for, it is not necessary to transmit the 15–bit address to that unit, but rather suffices to send the least significant 12 bits of the address to the 12–bit MA register in that unit. The external B registers, however, are copies of the B or MB registers in the users. Thus, if the Central Processor B register is holding a word to be stored in external memory, the word is transmitted to the B register of the addressed unit after decoding from MA indicates which unit is involved; conversely, if a word is read out from an external memory panel, it is held in the B register of that unit so that it can be written back into memory. The sense amplifiers send the word to the B register and at the same time transmit it on the line. Here it is double inverted before arriving at the user's B register memory. Parity is checked (on READ) and generated (on WRITE) in the user's internal B register.

These facilities make it possible for the MM–10's to operate quite independently once the memory operation has been initiated. The user supplies the address information, the type of operation, and a start signal. In WRITE operations, the word is supplied; in READ operations, the user waits for the word to be sent to the internal B register. In all cases, the memory cycle is completed under the direction of the timing counter and the unit involved will not be available until the cycle is completed.

The basic timing counter is the same for every memory panel, whether internal or external. The counter requires six microseconds to complete a memory cycle, thus establishing the basic six micro—second memory cycle for both internal and external memory operations. This counter shown in Figure 11.1—1, is started on receipt of an MS (Memory Start) signal. Definitions for the signals generated by the timing counter are given below:

RM: Read Memory signal; this signal is high for the first half of the memory cycle (when the information stored in the addressed location is read out);

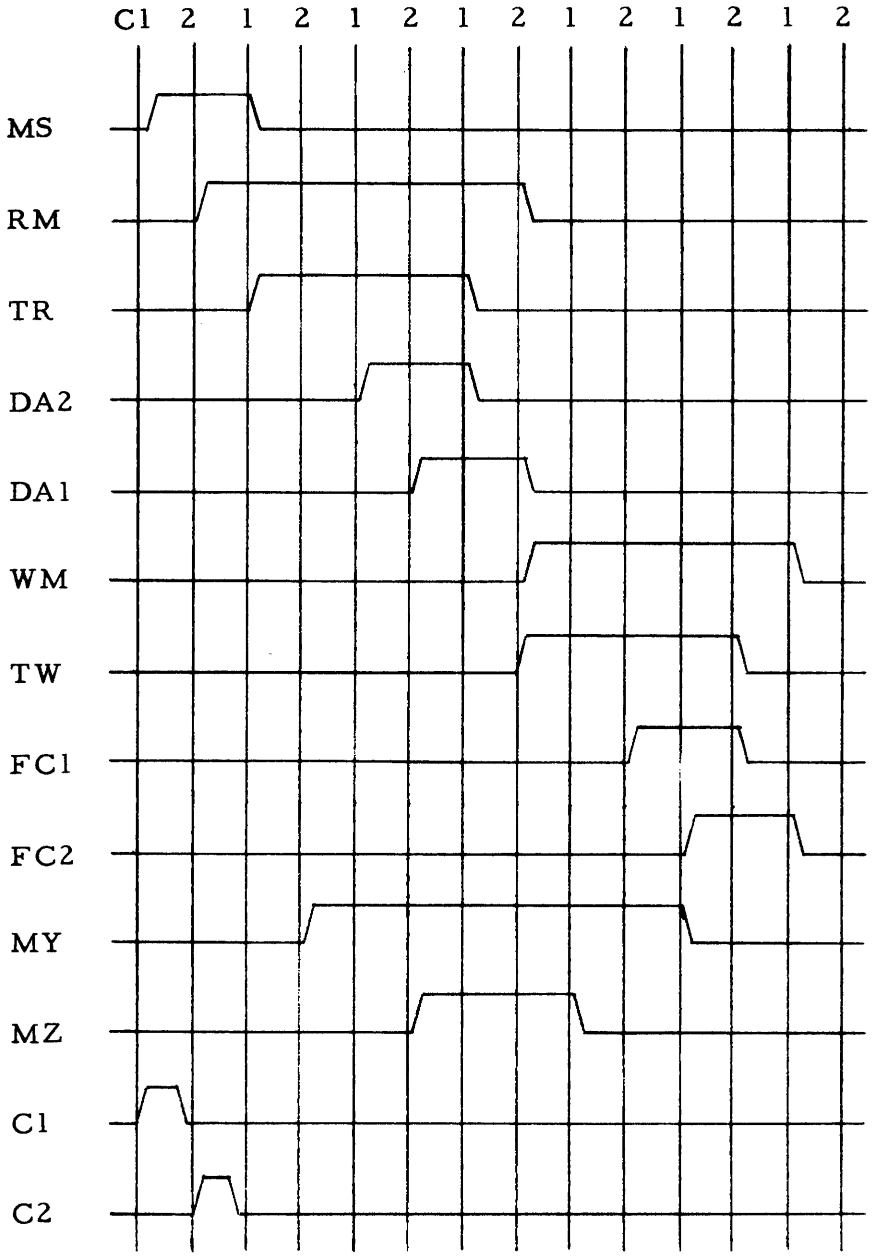
TR: Time of Read signal; this signal is high for the time in the first half of the memory cycle during which reading of the contents of the location is actually under way;

DA: Data Available signal; DA is high to indicate that the read operation has been completed; (DA sets the DAS and DAR flip—flops – see below);

WM: Write Memory signal; this signal is high for the second half of the memory cycle (when the information from register B is stored in the addressed location);

TW: Time of Write signal; this signal is high for the time in the second half of the memory cycle during which the writing of the contents of register B into the addressed location is actually under way;

FIGURE 11.1–1 Basic Memory Timing Counter



FC: Finish Cycle signal; this signal is high when the memory cycle is complete; it is used by the QM memory sequencer to gate the return to idle;

MY,
MZ: Timing signals internal to the memory timing counter and not used elsewhere.

It is obvious from this that the cycle has two parts: Read and Write. During read, the information from the addressed location is sensed and, due to the destructive–read characteristic of core memory, the location is zeroed. During write, the information stored in the associated B register is transferred into the addressed location.

Communication between the addressed timing counter and the memory user is kept to a minimum. The timing counter receives and sends the essential control signals on the control bus and the address information on the address bus. The word being handled is transmitted or received on the write or read bus. This communication between the MM–10 and the memory user is the same in all cases; the MM–10 does not distinguish between users. Aside from this, the logic within the user operates independently, relying in WRITE operations, on the sending of the Memory Start signal (MS) to the addressed timers and, in READ operations, on a timing signal from the addressed unit indicating transmission of the word read out in order to synchronize its own activities with the progress of the actual memory cycle. Thus, the memory user detects the arrival of the information in its own B register from the results of its own logic rather than from a signal from the timing counter. This mode of operation is desirable for several reasons. It keeps to a minimum possible timing difficulties on the bus lines, cuts down on communication between the units which is expensive in time and money, and allows flexibility in use of the basic cycle. Consider the memory operations performed by the Central

Processor using this memory cycle.

Before a WRITE operation is initialized, the word to be stored is sent to the internal B register. The WR flip–flop (WRite) is set to inhibit sensing of the core outputs. Thus, the Read portion of the memory cycle is used to clear the location to zero. This is necessary since the transfer from register B into memory is single–sided. The Write part of the cycle then stores the word from B into the addressed location.

A READ operation necessitates the saving of the word read out. When WR is low, the sensed information is sent to B. However, this is also a single–sided transfer, so the B register is cleared before a READ operation is started. The G–20 sequencer that requested the information is informed of its arrival in the internal B register by means of a DAS signal (Data Available Signal). The sequencer waits for this DAS, then proceeds with its own operation while the timing counter completes the memory cycle by restoring the word from the external B register back into the cleared location.

The DELAY–WRITE mode of operation amounts to a modified form of WRITE. The addressed location is read out into B at which point the timing cycle is halted. The sequencer calling for the operation receives the DAS signal and proceeds to send the new word to register B, thereby erasing the word just read out. The sequencer signals the memory timing counter to complete the Write portion of the cycle, and the word is correctly stored. The provision for saving the contents of the location and then deliberately destroying the word seems nonsensical unless viewed historically. DELAY–WRITE was included in order to implement the EXCHANGE command. This command called for the exchange of a word in memory with one stored in another device and, thus, necessitated the saving of the word read out until it had been picked up from register B, and the halting of the memory cycle until the new word was available in B. This command no longer exists, but the provision to save the word remains in the DELAY-WRITE mode. This mode is, in fact, used only by the KW sequencer which controls input/output operations on the Central Processor Communication Line. When a block of information is being received, KW calls for the reading of the location to be used for storing the next word while that word is being received. The DAS signal causes KW to send the new word to B, following which KW sends an MF signal (Memory Finish) to start completion of the memory cycle.

MM–10 control logic is contained in each memory user and varies somewhat from unit to unit. This logic must, among other things, make the following determinations:

1) is the addressed memory panel internal or external? (A DC–11 unit will know this already; it has no internal panels);

2) if the panel is external, which memory module (MM–10) is involved?

3) is the addressed MM–10 available?

4) is the address bus free?

The method employed by the user to make these determinations will be similar in all units; the handling of the bus system will vary from unit to unit. In this manual, it is only appropriate to discuss the way in which the Central Processor makes these decisions.

The flip–flops in the Central Processor that decode this information use the following signals to do so:

1) the Gate Data signal (GDA for unit A, GDB, GDC) is used during READ operations to indicate the timing of the actual read out of information and the transmission of the word to the

Central Processor;

2) the Finish Cycle signal (AFC for unit A, BFC, CFC), sent by the timing counter in operation when the 6 microsecond cycle is completed, is used in determining the availability of external units for future operations;

3) signals resulting from decoding of bits 14MA12 are used to determine which unit is addressed;

4) signals indicating bus line availability.

Availability decoding for external memory units amounts to a determination of the addressed panel in conjunction with an examination of the current status of the panels in that unit. Bits 14MA12 identify the addressed panel. These panels are lettered A, B, C, D, E, F, G, H (assuming an eight–panel system) with panel A containing the lowest numbered addresses. A decoding exists for the selection of each of these panels; thus, if 14MA12 contains 000, the SMA (Select Memory A) decoding is high. The MM–10 units are, confusingly enough, also lettered A, B, and C with unit A containing the addresses following those in the Central Processor, etc. In the usual case, the Central Processor will contain panels A and B and each MM–10 will contain two panels so that unit A will contain panels C and D, unit B, panels E and F, etc. The schematics represent this scheme of things. A slight bit of rewiring makes possible the change to a system wherein the Central Processor or MM–10 contains a single panel.

Availability of panels A and B (internal memory) need not be analyzed in this manner; a memory operation cannot be requested unless the Central Processor sequencers which handle memory operations are available, and they are never available if internal memory is still in operation. External units, however, can be addressed by other users or, in the case of WRITE operations, the unit may be completing a cycle that the Central Processor believes to have been finished a couple of microseconds earlier. (This handling of external WRITE operations is discussed in detail later.) Thus, it is quite possible for these units to be tied up when the Central Processor requests access. This availability is determined by use of the Memory Ready flip–flops. Each MM–10 has a corresponding MR flip–flop, these being MRA, MRB, and MRC for units A, B, and C respectively. When a start signal is sent to unit A, the MRA flip–flop is reset; when unit A finishes its cycle and transmits the Finish Cycle signal (FCA for unit A), the MRA flip–flop is set. Thus, if memory unit A is addressed, availability is determined by means of the MRA signal. The External memory ReaDy signal (ERD), which starts the external memory cycle, is high when the addressed unit and the address bus are free. When ERD goes high it causes transmission of a Memory Start signal (MS) to the addressed unit; both timing counters in the unit are started, but only the addressed one remains in operation. (Signal ERD has an additional qualifying condition for DELAY–WRITE operations: it does not come high until the input/output sequencer stores the received word in register B; following this, the memory operation is handled as a normal WRITE operation and signal ERD comes high when the addressed unit and the address bus are free.)

Central Processor memory logic is handled by two sequencers, QM and QB. These communicate with other sequencers through use of the following signals:

1) RQM;
2) GCA or GBA;
3) DAS;
4) MF;

1) The RQM signal (READY QM sequencer) indicates that the

memory logic is available for use. No request for memory operations can occur unless RQM is high; the user sequencer idles for RQM, then sends a GCA or GBA signal. The distinction between the two cases is that, for GCA, the designated address is stored in register CA and contains either the address of the next command in the program or the next operand in the block (for block input/output or repeat logic and arithmetic operations) while for GBA the BA register holds either the assembled operand which is to be used as an address, an index address, or (A) from the command word.

2) When the GCA or GBA signal is sent, the internal B register is cleared for READ and DELAY–WRITE operations. (For WRITE, the word to be stored is already in B.) Transmission of GCA or GBA has the following results:

a) the address is sent to the internal MA register for decoding;

b) for WRITE operations, set the WR flip–flop; for DELAY–WRITE, set the DW flip–flop;

c) GCA causes GQM to be sent; GBA causes SQM to be sent;

d) the address is incremented by 1 via the AC increment path. In the case of GCA, this leaves in CA the address of the next command in the program or the next operand in the block; in the case of BA, this is necessary since the information being stored or accessed may be double precision and, thus, call for use of the next address.

e) an internal MS signal is generated. QB is started simultaneously with QM by means of this internal MS. (QM is started by the SQM or GQM signal.) This signal also starts the internal memory counters. Decoding of 14MA12 will cause either QB or the internal timers to go out of operation. If the operation is internal, only one of the timers will remain in effect. If the operation is external, QB idles until it receives the ERD signal. External timing counters are not started until ERD is high. Recall that this signal indicates availability of the addressed unit and the address bus. In the case of DELAY–WRITE external memory operations, it also indicates that the input/output sequencer has sent the received word to register B and that the operation can proceed as a normal WRITE operation. When ERD is high, QB proceeds from its idle loop, the addressed MM–10 receives the necessary address information, and the external memory timing counter receives the external MS signal, MWR if the internal WR flip–flop is set, and the external B register of the addressed unit is cleared. The external MS starts both timing counters if the unit contains two panels, but only the addressed one remains in operation.

3) The DAS signal is sent to the user sequencer to indicate that the desired information is available in register B and can be picked up. The user sequencer waits for this signal only if the operation is READ or DELAY–WRITE. In READ, that sequencer will idle for the DAS signal and then proceed through its own cycle. (If the next operation called for is a memory operation, it will idle for the RQM signal.) In DELAY–WRITE, the input/output sequencer waits for DAS to signal the arrival of the word read out in the B register before it sends the received word to the B register to be stored.

4) The user sequencer then sends the MF signal to indicate that the WRITE operation may proceed. (This is handled differently for internal and external operations. For internal memory

operations, the signal MF goes directly to the timing counter and starts the Write half of the memory cycle. For external operations, the necessary DAS is generated within the G–20 before the addressed timing counter is started; the MF is then a qualifying term of ERD which will go high when the addressed external unit is available, the data is in internal B, and the address bus is free. The operation then proceeds as a normal WRITE. This is possible because the word read out and the word to be stored are held in two different B registers on external operations. For the internal DELAY–WRITE, the word to be stored would be destroyed if it were sent to the B register prior to the read out.

The logic involved in the handling of internal memory operations is much simpler than that involved in external operations. Internal memory is available only to the Central Processor. It is controlled directly by the QM sequencer. QM relies upon two signals, DAR and FC, to gate it from its two idle loops. The DAR signal represents the end of the READ half of the internal memory cycle and is sent by the DAR flip–flop when a DA signal is received from the addressed memory timing counter. (DA also sets the DAS flip–flop during internal memory operations.) The Finish Cycle signal, FC, is received directly from the internal timer on internal operations and gates QM back to the READY state. Since the cycle requires 6 microseconds to complete, all internal memory operations require 6 microseconds of Central Processor time.

External memory operations are traced internally by the QB sequencer. QM (which is started simultaneously with QB) is gated through its cycle by means of signals from QB so that memory logic availability will be signalled (RQM high) at the right time. It was pointed out earlier that communication between MM–10's and the memory user (DC–11 or G–20) is kept to a minimum. Once the external memory operation is initiated, QB carries the responsibility of reporting progress in the cycle to the other sequencers. Thus, QB sets the DAS and DAR flip–flops by means of the DAX signal. The setting of DAR always gates QM from the first idle loop. The DAS signal is sometimes useful, sometimes not:

1) on WRITE operations, the user sequencer does not require the DAS signal; the DAR signal is generated for the purpose of gating QM from the first idle loop.

2) On DELAY–WRITE, the DAS signal is generated before the memory cycle begins to cause the input/output sequencer to send the received word to the internal B register. The DAR generated at the same time will gate QM. (These are set redundantly in DELAY–WRITE after the memory cycle starts since DELAY–WRITE is handled at that time as a WRITE operation.)

3) On READ operations, the user sequencer waits for the DAS signal to indicate that the word is available in the internal B register; thus, setting of DAS is dependent upon the actual transmission of the word read out; i.e., it is set two micro–seconds after the Gate Data signal indicates that the word is being read and transmitted. The DAR signal is also set to gate QM from its idle loop.

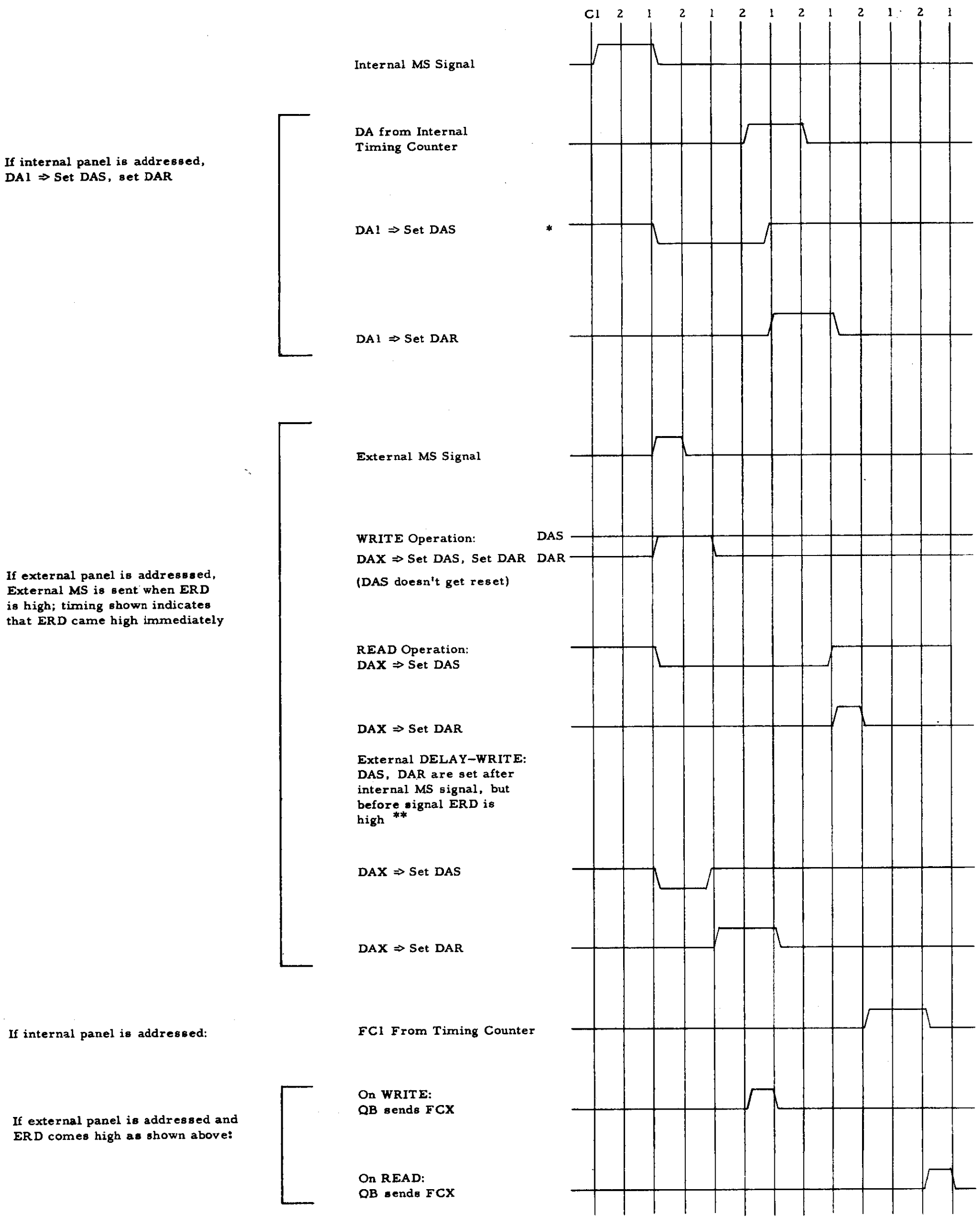
QB also generates a pseudo–Finish Cycle signal to gate QM back to the READY state. This signal (FCX) sets the FC flip–flop three micro–seconds before the actual end of the memory cycle on external WRITE operations. This effectively shortens the Central Processor cycle time and, since the memory logic is made available, allows for immediate use of the memory if it doesn't involve the unit which is still finishing up.

Figure 11.1–2 demonstrates these timing distinctions for internal operation, the sequence of events is always the same; for external operations, timing varies.

The QB sequencer has the capacity to serve as intermediary between the MM–10 and the other internal logic because QB's operation is synchronized with that of the addressed timing counter. Recall that QB idles, waiting for signal ERD, until the addressed unit is available and the address bus is free. ERD high sets in motion both the addressed timing counter (via the external MS signal) and the QB logic (which is gated directly by signal ERD. For WRITE operations, it remains only for the word stored in the internal B register to be gated to the external unit when the WRITE bus is free, i.e., two microseconds after the ERD signal is high. This bus will always be available when needed since timing requirements on WRITE operations do not vary, i.e., the address bus availability used to start the memory cycle implies write bus availability at the appropriate time interval.

On READ operations, the problem is more complex since it is necessary to provide for the case where a slower or faster memory unit is added to the system. No such unit is now used with the system, but it is anticipated that there may be a need for such provisions. Thus, the time interval between the arrival of the ERD signal and the transmission of the word read out to the internal B register cannot be predicted. The signal sent by the MM–10 to indicate this transmission is the Gate Data signal (GDA for unit A, GDB, GDC). This signal is used by the QB sequencer to set the DAL flip–flop. An idle loop provides for the case where DAL is low. (This loop will be taken for slower memory devices, not for MM–10's.) The Gate Data signal, the FC signal from the timing counter (used to qualify the ERD signal), and the transmission of the word on the read bus represent the only communication between QB and the MM–10 during a memory operation

FIGURE 11.1–2 Timing Diagram for Signals DAS, DAR and FC


C1 2 1 2 1 2 1 2 1 2 1 2 1
Internal MS Signal
If internal panel is addressed, DA1 ⇒ Set DAS, set DAR
DA from Internal Timing Counter
DA1 ⇒ Set DAS
*
DA1 ⇒ Set DAR
External MS Signal
If external panel is addressed, External MS is sent when ERD is high; timing shown indicates that ERD came high immediately
WRITE Operation:
DAX ⇒ Set DAS, Set DAR
(DAS doesn't get reset)
DAS
DAR
READ Operation:
DAX ⇒ Set DAS
DAX ⇒ Set DAR
External DELAY–WRITE: DAS, DAR are set after internal MS signal, but before signal ERD is high **
DAX ⇒ Set DAS
DAX ⇒ Set DAR
If internal panel is addressed:
FC1 From Timing Counter
If external panel is addressed and ERD comes high as shown above:
On WRITE: QB sends FCX
On READ: QB sends FCX

*The DAS flip–flop is in the set condition when QM is started and is reset on the second clock (except in external WRITE).

**Following receipt of MF signal from KW, indicating that the received word is stored in the internal B register, the DELAY–WRITE operation is handled as a normal WRITE operation waiting for ERD.

after the ERD signal goes high (with its concomitant transfers of address information, etc.).

The operations of the QM and QB sequencers have been described in some detail in this section because their activities are so interdependent. Table 11.1–1 is included to demonstrate the similarities and the distinctions between the various operations. Details of implementation are covered in Section 11.2 which includes the algorithms and flow charts for each.

TABLE 11.1–1 Memory Users

	REGISTERS SIGNALS PATHS UNDER DISCUSSION	WRITE		READ				DELAY–WRITE
		KP	KJ	KC (Command access and repeat)	KC (Operand assembly)	QM	KW	KW
Set up for memory operation	Internal B register	Word to be stored must be sent to B		Contains last word accessed or stored; not cleared at this point.				
	External B registers	Contains last word stored or accessed; not cleared at this point.						
	Current address sent to address register	BA	CA	CA	BA	CA	CA	CA
Start memory operation	SEQUENCER REQUIRING MEMORY CYCLE MUST WAIT FOR SIGNAL RQM INDICATING THAT QM IS AVAILABLE BEFORE STARTING OPERATION							
	Starting signal	GBA	GCA	GCA	GBA	GCA	GCA	GCA
	Signals generated: START to QM	SQM	GQM	GQM	SQM	GQM	GQM	GOM
	Mode of operation:	WR	WR					DW
Start internal* memory timers		MS*	MS*	Internal B register is cleared MS*	MS*	MS*	MS*	Internal B register is cleared MS*
	Actions taken as START signal is sent to QM:	MS goes to internal timing counters; an internal timer remaining in operation only if addressed; otherwise it drops out. MS also starts QB; if decoding from MA indicates an internal operation, QB goes to idle.						
**Start memory timers in addressed MM–10 unit	Paths enabled by starting signal:	BA(0)AC BA(0)MA	CA(0)AC CA(0)MA	CA(0)AC CA(0)MA	BA(0)AC BA(0)MA	CA(0)AC CA(0)MA	CA(0)AC CA(0)MA	CA(0)AC CA(0)MA
		(This is the internal MA register only.)						
	EXT (QB only)	This signal comes high if decoding of MS shows that external memory has been addressed; if it is low, QB returns immediately to idle.						
	ERD (QB only)	This signal gates the next step in QB; it is high when the addressed MM–10 and the address bus are free.						On DW, ERD is always low the first time through. DW causes generation of false DAS and DAR signals so that input/output control will store the received word in internal B register.
								When this is done, input/output supplies MF which is, in this case, necessary to cause ERD to go high. Thus, the bus, MM–10 and MF must all be available before OW proceeds
	Action for ERD: (QB only)	Least significant 13 bits of address are sent to MM–10: 11MA0 to external MA, MA12 to select memory timer. Timing counters in addressed MM–10 started by MS**, only the addressed counter will remain in operation. Register B of addressed unit is cleared.						
	Mode of operation: (QB only)	MWR	MWR	REM	REM	REM	REM	MWR (From this point on, delay–write is handled as a WRITE operation by QB)
RM: Read Mode, 1st half of basic memory cycle; ends with generation of DAS, DAR		INTERNAL MEMORY: DA signal from memory timing counter indicates that the location is cleared and the write part of the cycle may proceed; DA sets the DAS and DAR flip–flops; DAR gates QM from idle. EXTERNAL MEMORY: DAX signal from QB sets the DAS and DAR flip–flops; DAR gates QM from idle; this is done at the start of the memory cycle to enable early exit from QM.		INTERNAL MEMORY: DA signal from memory timing counter indicates that the data is available in the internal B register; DA sets the DAS and DAR flip–flops to gate QM from idle and signal user that it can pick up the data. EXTERNAL MEMORY: DAX signal from the QB sequencer sets the DAS and DAR flip–flops; DAX is high 2 us after receipt of the Gate Data signal from the addressed unit that signals transmission of the word to the internal B register; DAS and DAR are used as described above.				INTERNAL MEMORY: DA signal from memory timing counter indicates that the data is available in the internal B register; DA sets the DAS and DAR flip–flops; input/output sequencer waits for DAS, then sends received word to internal B register, overlaying the word read out; DAR gates QM from idle.
WR: Write Mode, 2nd half of basic memory cycle; ends with generation of FC	FC (FCX for external memory) End of WRITE portion of memory cycle	INTERNAL MEMORY: signal from memory timing counter gates QM to READY. EXTERNAL MEMORY: QM doesn't need to wait for this signal from external timers; QB supplies pseudo–finish FCX to advance QM to READY; QB goes to idle. Total time from start of memory timers: 3 micro–seconds, another memory operation can be started immediately if it doesn't involve the same MM–10. Actual FC generated by external timer is used in Central Processor to determine the availability of that unit.		INTERNAL MEMORY: signal from memory timing counter gates QM to READY. EXTERNAL MEMORY: again, pseudo–finish is sent to QM, this time only 1 microsecond early; QM to READY, QB to idle. Actual FC generated by external timer is used in Central Processor to determine the availability of that unit.				INTERNAL MEMORY and EXTERNAL MEMORY: identical to WRITE operation.

SECTION 11.2 – THE QM AND QB SEQUENCERS

In comparison with the discussion of the memory system in general, the description of the operation of the QM sequencer is remarkably simple. This is true because QM does not take cognizance of the operation being performed, i. e., it operates in the same way whether the operation is internal or external, and is unaware of the state of control signal WR (high for WRITE operations) or signal DW (high for DELAY–WRITE operations). These signals are sent to the timing counters, not to QM.

QM has the following tasks:

1) it checks the designated address; if the address does not exist in the machine, MNA will be high causing QM to send an interrupt request and exit;

2) it increments the current address by 1;

3) it idles until it receives a DAR signal; DAR can be set by a DA signal from the internal timers or by a DAX signal from the QB sequencer. If neither of these terms is available, DAR and DAS are reset. This is necessary since DAS will still be set from the preceding memory operation and must be reset so that the user sequencer will not read a false signal;

4) if a memory parity error is found on a READ operation, it hangs up in an idle loop, causing the Memory Parity light to go on in the Central Processor control panel. (Memory parity errors will not occur on WRITE operations since the parity is generated, not checked, during WRITE.)

5) if no memory parity error occurs, QM idles until an FC signal arrives; the FC flip–flop is set by the internal timing

counter at the end of the internal cycle, or by the FCX pseudo–finish signal from the QB sequencer as soon as the current operation is no longer dependent upon Central Processor control;

6) it signals READY QM sequencer, indicating memory logic availability, when gated back to its normal idle loop.

The two branches in QM (GQM and SQM) correspond to the two start signals GCA and GBA and differ only in that, for GCA the address is stored in register CA while for GBA the address is in register BA. Otherwise, the branches are identical. (See Section 11.1 for a discussion of what happens when GCA or GBA is sent.)

Operation of the QB sequencer has also been described in some detail. The QB basic algorithm is shown in Figure 11.2–3 and the flow chart in Figure 11.2–4. For purposes of clarity, it might be well to review the following points:

1) QB is started by the sending of the internal MS signal; this signal also starts the internal timing counters; if decoding of 14MA12 causes SMA or SMB to go high, the EXT (EXternal Memory) signal is low and QB returns to idle; otherwise, if SMC, SMD, SME, SMF, SMG or SMH is high, EXT is high and QB remains in operation while the internal timing counters go out. Thus, QB is active only on external memory cycles.

2) ERD governs the actual start of external memory cycles; for WRITE and READ operations, ERD is high when the addressed memory is available and the address bus is free; for DELAY–WRITE operations, there is an additional qualifying term. QB generates (at QBC) an early DAS signal which informs the input/output sequencer that it can proceed

to store the received word in the internal B register. When it has done this, it sends signal MF. On external DELAY–WRITE operations, MF is necessary to qualify ERD. During the actual memory cycle, DELAY–WRITE can thus be handled as a simple WRITE operation.

3) The memory timers of the addressed unit are actually started at QBD by sending of an external MS signal when ERD goes high; only the addressed timer will remain in operation, but both are started.

4) Flip–flop DAL (DAta on Line) goes high on READ operations when the Gate Data signal indicates that the word read out is being transmitted on the read bus. Timing of the MM–10 is such that this signal will always be high at QBF when that unit is accessed and the idle loop will not be taken. For slower memory devices, this idle loop would be used while waiting for the data to be available.

5) Flip–flop DAT (DAta) is set to gate the word to be stored from the internal B register to the external memory via the B(T)EM path. (This transfer takes place on the write bus.)

The effective WRITE cycle (the time during which QM is not READY) is shortened to three microseconds on external operations by sending of an early DAR signal and an early pseudo–finish signal to QM. It should be noted that, on READ operations, the DAS flip–flop is high after three microseconds during internal operations and after four microseconds during external operations.

FIGURE 11.2–1 Algorithm for QM Sequencer

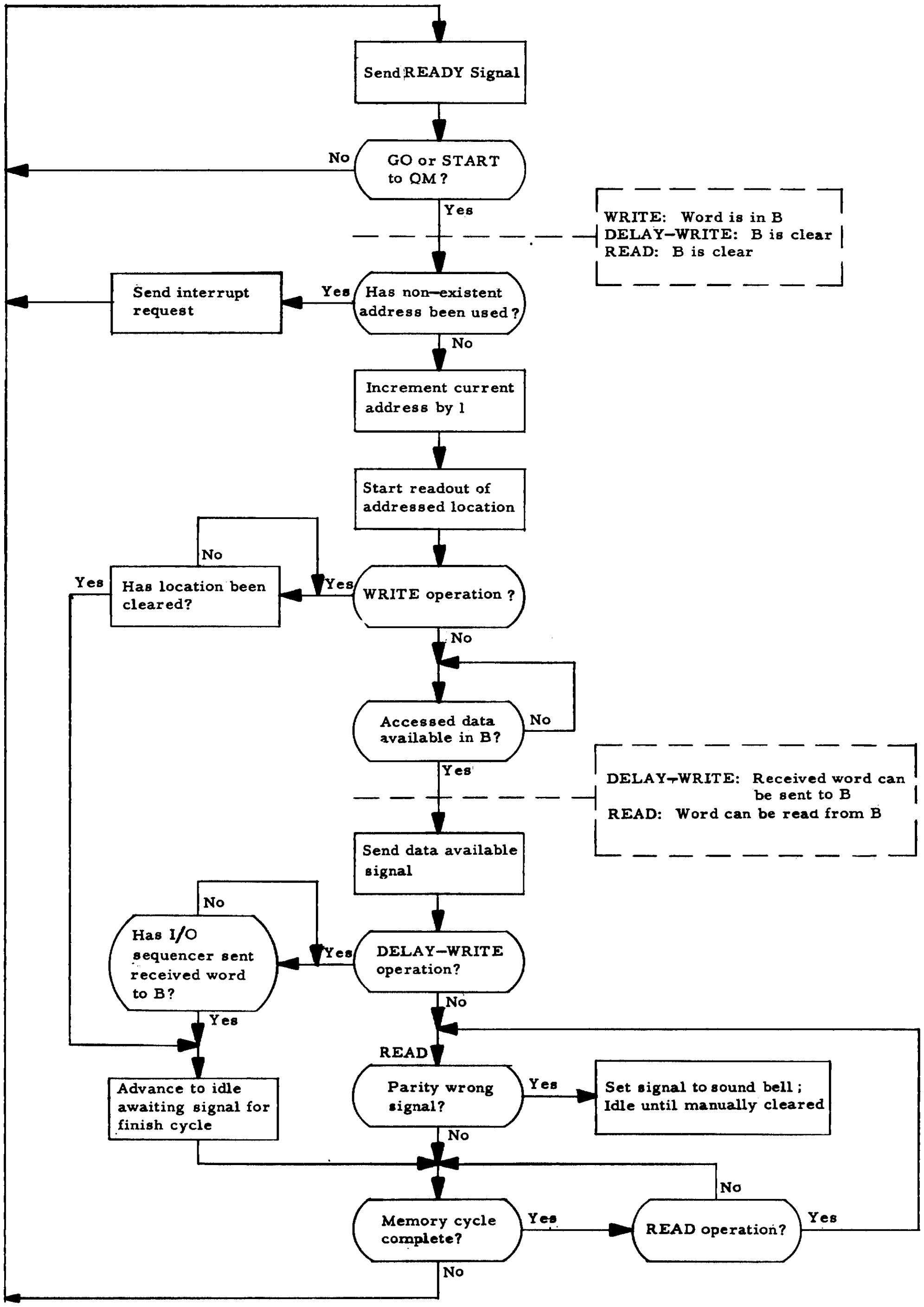

Send READY Signal
GO or START to QM?
No
Yes
WRITE: Word is in B
DELAY–WRITE: B is clear
READ: B is clear
Send interrupt request
Yes
Has non-existent address been used?
No
Increment current address by 1
Start readout of addressed location
No
Yes
Has location been cleared?
Yes
WRITE operation ?
No
Accessed data available in B?
No
Yes
DELAY–WRITE: Received word can be sent to B
READ: Word can be read from B
Send data available signal
No
Has I/O sequencer sent received word to B?
Yes
DELAY–WRITE operation?
Yes
No
READ
Parity wrong signal?
Yes
Set signal to sound bell; Idle until manually cleared
Advance to idle awaiting signal for finish cycle
No
No
Memory cycle complete?
Yes
READ operation?
Yes
No

TABLE 11.2–1 Terms Used on the QM Flow Chart

DA	DAta available signal from addressed timing counter; on internal memory operations, DA sets flip–flops DAS and DAR.
DW	DELAY–WRITE flip–flop; DW is set by the user sequencer (KW sequencer is the only user of DELAY–WRITE) at the time the start signal is sent to the memory logic.
FC	Finish Cycle flip–flop; FC is set by the addressed timing counter. The FC signal is sent by the addressed timing counter on internal memory operations; on external operations, the FC flip–flop is set by the pseudo–finish signal (FCX) generated by QB to gate QM back to READY.
Fix DAS, DAR	Data Available Signal and DAta Ready flip–flops; DAS and DAR are set by the DA signal on internal operations and by the DAX signal on external operations; Fix DAS, DAR is an abbreviation for $\mathrm{DA} \vee \mathrm{DAX} \Rightarrow \text{set DAS, set DAR}$ $\overline{\mathrm{DA}} \wedge \overline{\mathrm{DAX}} \Rightarrow \text{reset DAS, reset DAR}$ The resetting is necessary for the DAS flip–flop since it is left set at the end of a memory operation. It is left in a set condition because the user sequencer may not read it immediately. Thus, it must be cleared on the next memory operation before the next user sequencer reads it.
GQM	GO signal to QM sequencer; the sending of signal GCA generates signal GQM.
JNC	Indicates the sending of an interrupt request, an action that is taken when the designated address is found to be non–existent.
MNA	Memory Non–existent Address; MNA is high when the current address is found to be non–existent.
PW	Parity Wrong flip–flop; PW goes high on READ operations when a check on the parity bit shows it to be in the wrong state; this condition causes QM to hang up so that the machine must be initialized before the logic can be used again.
RQM	READY QM sequencer.
WR	WRite flip–flop; WR is set by the user sequencer when requesting a WRITE operation.

FIGURE 11.2–2 The QM Flow Chart

QM

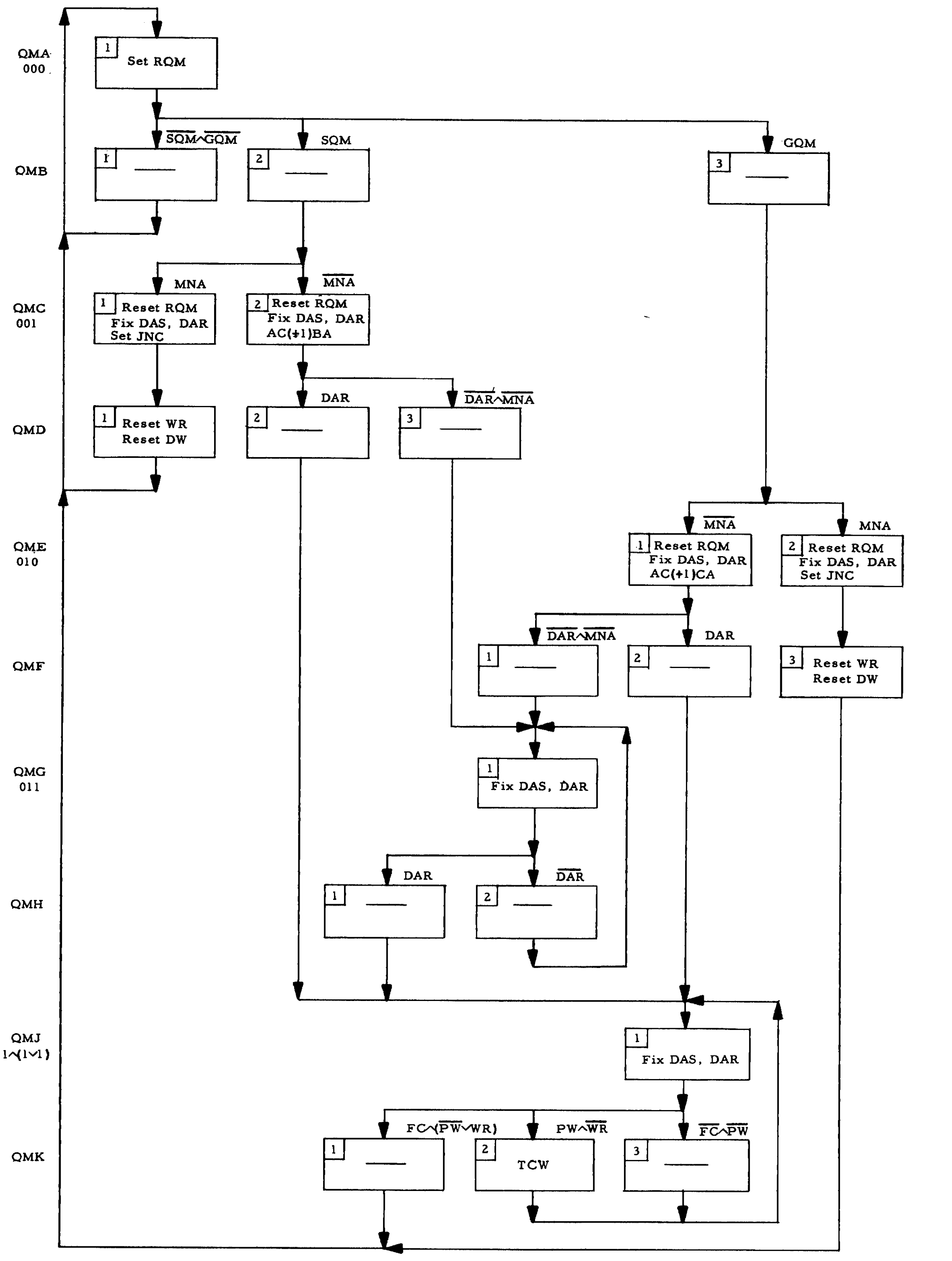
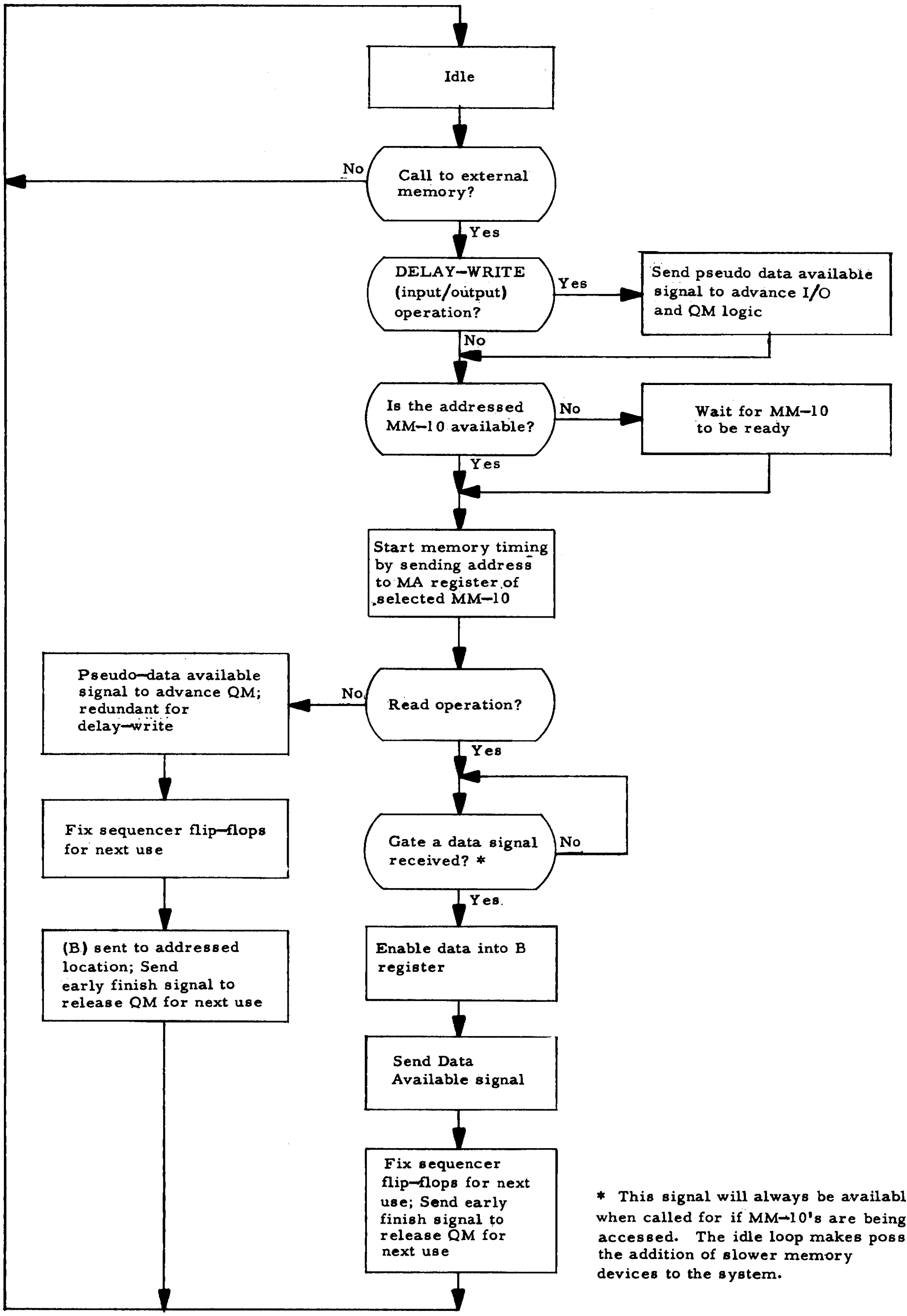

QMA
000
1 Set RQM
QMB
$\overline{SQM} \wedge \overline{GQM}$
SQM
GQM
1
2
3
QMC
001
MNA
$\overline{MNA}$
1 Reset RQM Fix DAS, DAR Set JNC
2 Reset RQM Fix DAS, DAR AC(+1)BA
QMD
1 Reset WR Reset DW
DAR
$\overline{DAR} \wedge \overline{MNA}$
2
3
QME
010
$\overline{MNA}$
MNA
1 Reset RQM Fix DAS, DAR AC(+1)CA
2 Reset RQM Fix DAS, DAR Set JNC
QMF
$\overline{DAR} \wedge \overline{MNA}$
DAR
1
2
3 Reset WR Reset DW
QMG
011
1 Fix DAS, DAR
QMH
DAR
$\overline{DAR}$
1
2
QMJ
1^(1v1)
1 Fix DAS, DAR
QMK
FC^($\overline{PW}$vWR)
PW^$\overline{WR}$
$\overline{FC}$^$\overline{PW}$
1
2 TCW
3

FIGURE 11.2–3 Algorithm for the QB Sequencer


Idle
Call to external memory?
No
Yes
DELAY–WRITE (input/output) operation?
Yes
Send pseudo data available signal to advance I/O and QM logic
No
Is the addressed MM–10 available?
No
Wait for MM–10 to be ready
Yes
Start memory timing by sending address to MA register of selected MM–10
Read operation?
No
Pseudo–data available signal to advance QM; redundant for delay–write
Fix sequencer flip–flops for next use
(B) sent to addressed location; Send early finish signal to release QM for next use
Yes
Gate a data signal received? *
No
Yes
Enable data into B register
Send Data Available signal
Fix sequencer flip–flops for next use; Send early finish signal to release QM for next use

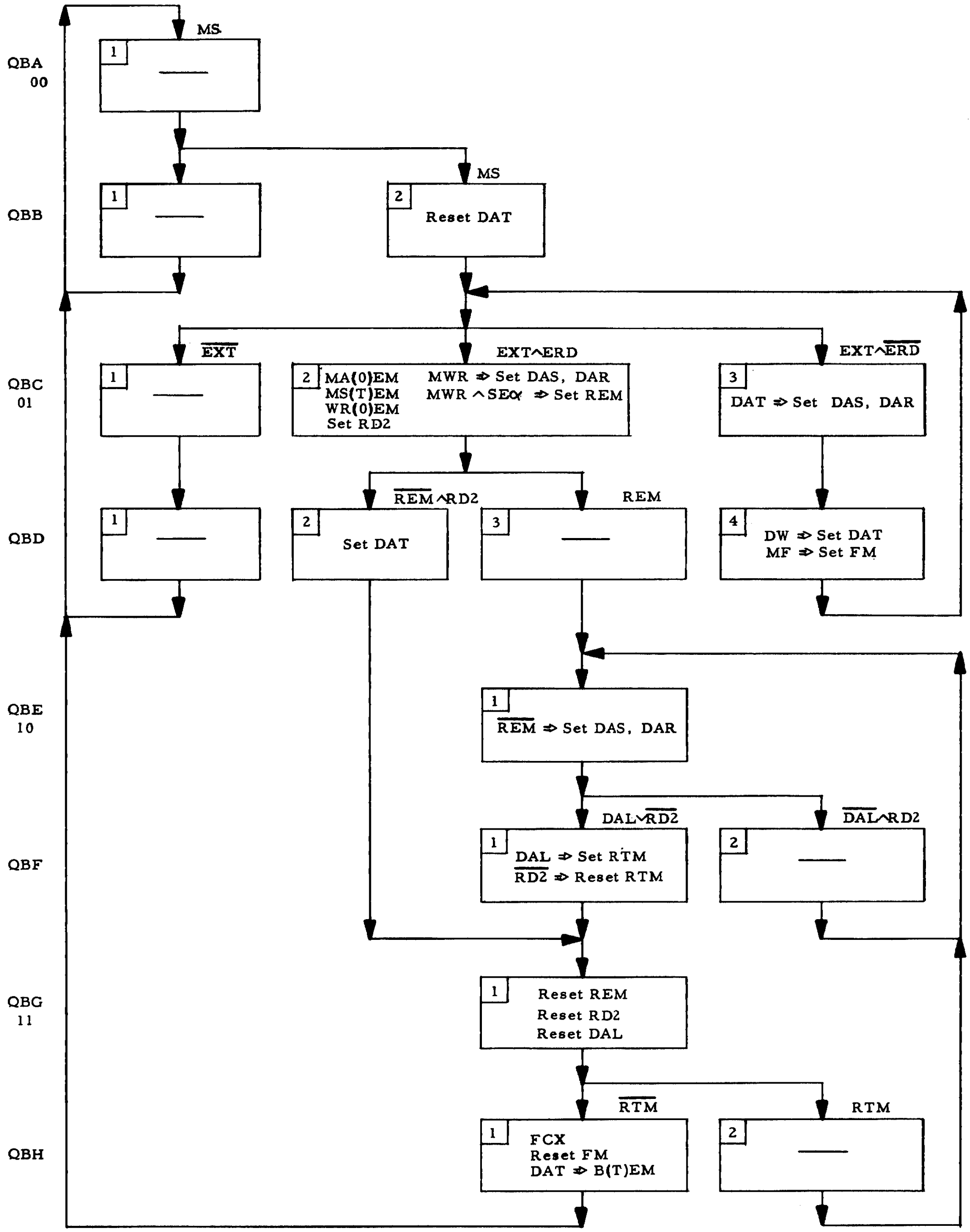
* This signal will always be available when called for if MM–10's are being accessed. The idle loop makes possible the addition of slower memory devices to the system.

TABLE 11.2–2 Terms Used on the QB Flow Chart

DAR	DAta Ready flip–flop; set by QB to gate QM sequencer.
DAS	Data Available Signal flip–flop; set by QB whenever DAR is set; on DELAY–WRITE operations, the DAS is generated early so that the input/output sequencer will send the received word to the internal register B; on READ operations, DAS is sent to the user sequencer two microseconds after the transmission of the word to the internal B register begins. On WRITE operations, the signal is not used.
DAL	Data Available on Line flip–flop; DAL is set by the Gate Data signal from the MM–10 on external READ operations to indicate the actual transmission of the word from the addressed external memory to the internal B register.
DAX	Data Available eXternal signal; DAX sets the DAR and DAS flip–flops during external memory operations.
DW	DELAY–WRITE flip–flop; this flip–flop is set by the input/output sequencer that uses the DELAY–WRITE mode at memory startup time.
ERD	External ReaDy signal; ERD indicates that the address bus is free and the addressed unit is available; on DELAY–WRITE operations, it also indicates that the received word has been stored in the internal B register and the operation may proceed as a normal WRITE operation.
EXT	EXTernal memory signal; signal EXT is set when 14MA12 decoding indicates that an external memory panel has been addressed.
FC2	Finish Cycle signal from the memory timing counter; FC2 high indicates that the memory cycle will be complete one clock time later.
FCA, FCB, FCC	Finish Cycle signals from the memory timing counters of units A, B, and C.
FCX	Pseudo–Finish Cycle signal sent by QB to gate QM to the READY state.
FM	Finish Memory operation flip–flop; FM is set by the MF signal generated by the input/output sequencer during DELAY–WRITE when the received word has been transferred to register B.
GDA, GDB, GDC	Gate Data signal sent during external READ operations when the word is being transmitted to the internal B register from unit A, B, or C respectively.
MS	Internal Memory Start signal; (external MS is sent at QBC via enabling of the MS(T)EM path).
MWR	Memory WRite; MWR is high for external WRITE operations.
RD	ReaDy flip–flop; RD is set when an external memory operation begins and serves a bookkeeping function after that.
REM	Read External Memory flip–flop; REM is set to distinguish a READ from a WRITE operation. (DELAY–WRITE is processed as a normal WRITE operation.)
RTM	Read TiMe flip–flop; RTM is high during transmission of the word to the Central Processor during READ operations.
WR	WRite flip–flop; set by the user sequencer at the start of a memory WRITE operation.

FIGURE 11.2–4 The QB Flow Chart

QB



SECTION 11.3 – THE KP SEQUENCER

The purpose of the KP sequencer is to supervise the storing of information as called for by one of the five store commands or as required by the eight index commands, all of which call for modification of the contents of an index register and, therefore, must be completed by a return of the information to that location. KP concerns itself primarily with the requirements of the particular opcode and with the startup of the QM sequencer to perform the WRITE operation.

The functioning of KP differs in the two cases since the information necessary to performance of the store is differently placed.

1) Store commands: the contents of the Accumulator are to be stored in the address specified by the assembled operand; thus, QM will expect to find the assembled operand in BA, the low order portion of the contents of the Accumulator, plus exponent information, sign bits, and length flag if double precision, in B.

2) Index commands: the assembled operand is to be stored in the index register specified in the index portion of the command word; thus, QM will expect to find the assembled operand in the B register, the index address in BA. No exponent information or length determination is necessary in the case of index commands since information is always stored by these commands in integer format.

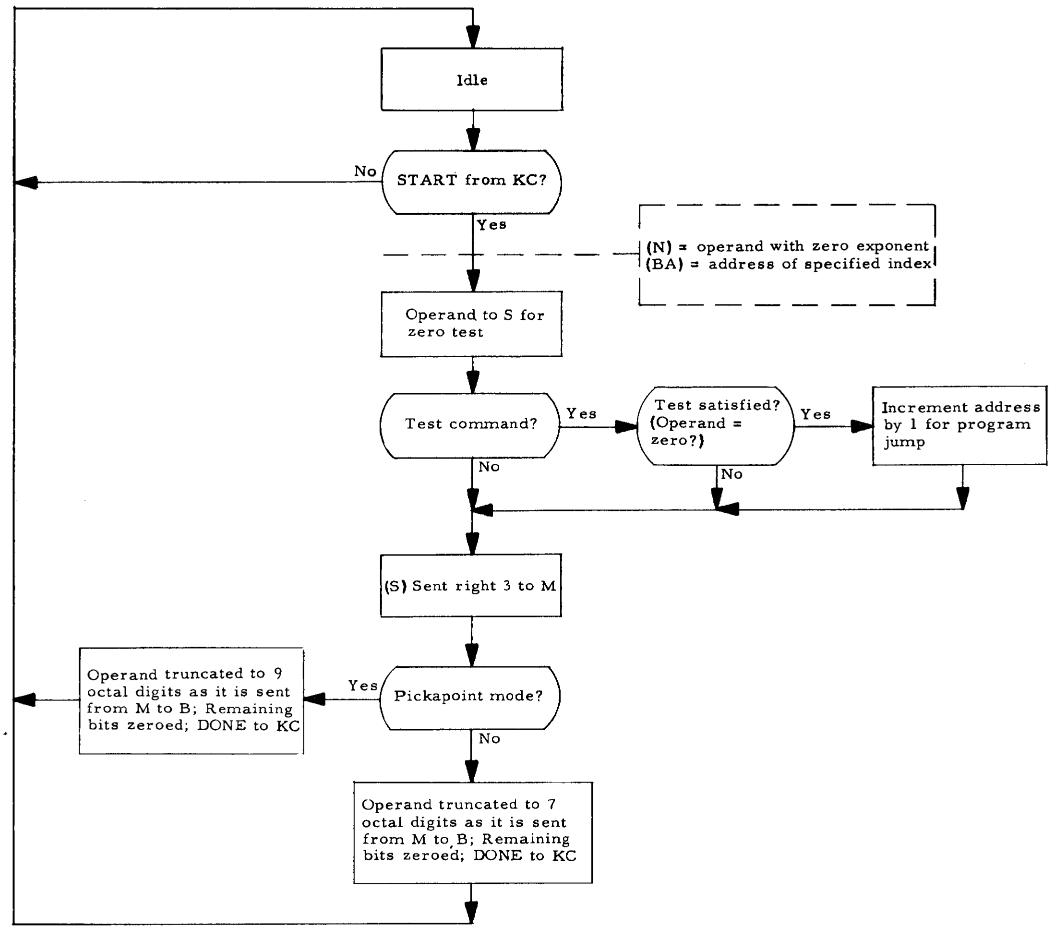
Two signals sent by Master Control start KP: GKP and SKP. SKP pertains only to the eight index commands. At the time this signal is sent, the decoding of the index commands will have been reduced to one of four (see description of this in the KC write–up). The signs of B1 and B5 will already have been reversed by KC. These commands are the same as the B0 and B4 commands, except that they call for loading the index negative rather than positive. Thus, with sign determination complete, there is no distinction between the B0 and Bl commands or B4 and B5. The only remaining distinction is between B0 and Bl which calls for loading the index register, and B4, B5 which calls for loading the register and testing the operand against zero. (In the test case, if the operand is zero, execution of the program will continue after one command is skipped.)

At the time SKP is sent, KC will have transferred the index address to the BA register, reduced the opcode decoding as indicated, assembled the operand, stored it in OA (N, EA, sign in WS), and shifted the operand to zero exponent since the storage must be in integer format. KP will send the operand to the B register, along with the sign from the WS flip–flop, as a seven or nine digit integer depending upon whether or not the pickapoint mode of storage is enabled. (Pickapoint integer format calls for a nine digit integer with zero exponent.) The transfer of the operand to B will effect a truncation of the integer to the desired length. Figure 11.3–1 demonstrates how this is handled; Figure 11.3–2 shows the relevant sequencer blocks.

At the time GKP is sent, Master Control will have assembled the operand which is to serve as the address for the store operation, shifted it to zero exponent and checked it for positive sign. This address is in OA at GKP. KP sends the address to BA, then formats the contents of the Accumulator in accordance with the store command, sends the operand to B, and starts QM.

These commands call for the storage of information in the Accumulator (register A), formatted according to the command specifications, in the address indicated by the assembled operand (which will be stored in BA) referred to here as X. The commands and their format

FIGURE 11.3–1 Final Processing of Index Commands by KP


Idle
START from KC?
No
Yes
(N) = operand with zero exponent
(BA) = address of specified index
Operand to S for zero test
Test command?
Yes
No
Test satisfied? (Operand = zero?)
Yes
No
Increment address by 1 for program jump
(S) Sent right 3 to M
Pickapoint mode?
Yes
No
Operand truncated to 9 octal digits as it is sent from M to B; Remaining bits zeroed; DONE to KC
Operand truncated to 7 octal digits as it is sent from M to B; Remaining bits zeroed; DONE to KC

requirements are discussed first, then the KP algorithm for carrying out these actions. The order of the discussion corresponds to the degree of complexity inherent in the processing of each. Thus, P2 and P4 are first, followed by P0 and P3, leaving P1 until last since it is the most challenging of the lot.

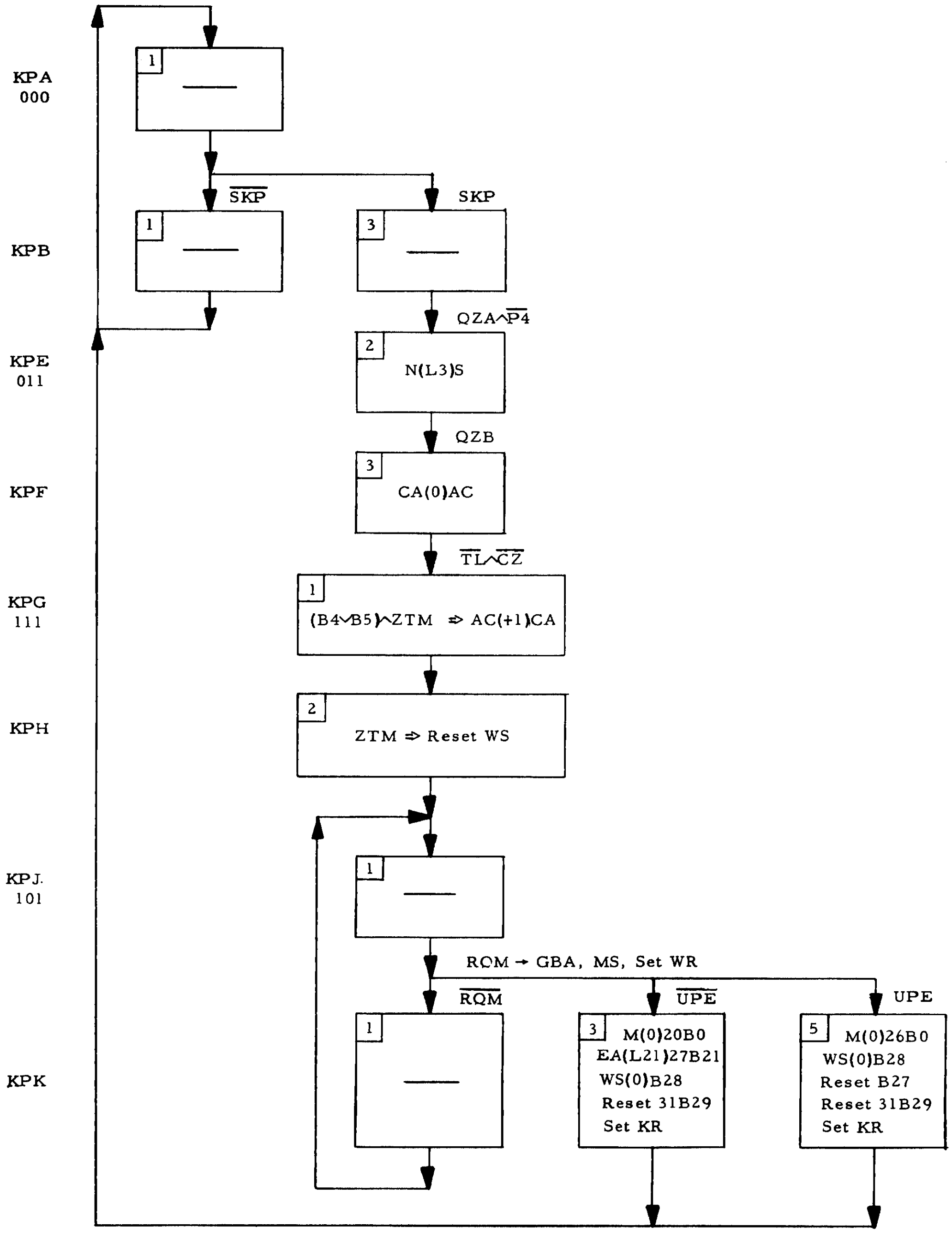
P2 (STL) – store logical – provides for the storage of the least significant 32 bits of the Accumulator following shifting of the contents

TABLE 11.3–1 Terms Used on the KP Flow Chart for Final Processing of Index Commands

B0, B1, B4, B5	Index opcodes LXP, LXM, XPT, XMT. At opcode startup, the decoding of the other four index commands will have been modified to agree with one of these commands so that only four are gated through KP (see Section 9.7).
Clear EA	Signal that clears exponent register EA to conform to zero operand.
KR	Opcode DONE signal to KC.
RQM	READY QM sequencer.
SKP	START signal to the KP sequencer for index opcodes.
WS	Working Sign flip–flop.
ZTM	Zeros To Memory signal; when high, ZTM indicates that zeros exist in the least significant seven (for non–pickapoint mode) or nine (for pickapoint mode) octals of the operand that is to be stored in the memory.

FIGURE 11.3–2 The KP Flow Chart for the Final Processing of Index Commands by K

KP



of the Accumulator to zero exponent.

$$31(A)0 \rightarrow 31(X)0$$

P4 (STZ) – store zeros – provides for zeroing a location; the contents of the Accumulator have no affect on this store operation.

$$0 \rightarrow 31(X)0$$

Figure 11.3–3 contains the basic algorithm for these commands, Figure 11.3–4 shows the corresponding sections of KP.

FIGURE 11.3–3 Algorithm for the Processing of Store Logical and Store Zero Commands by KP

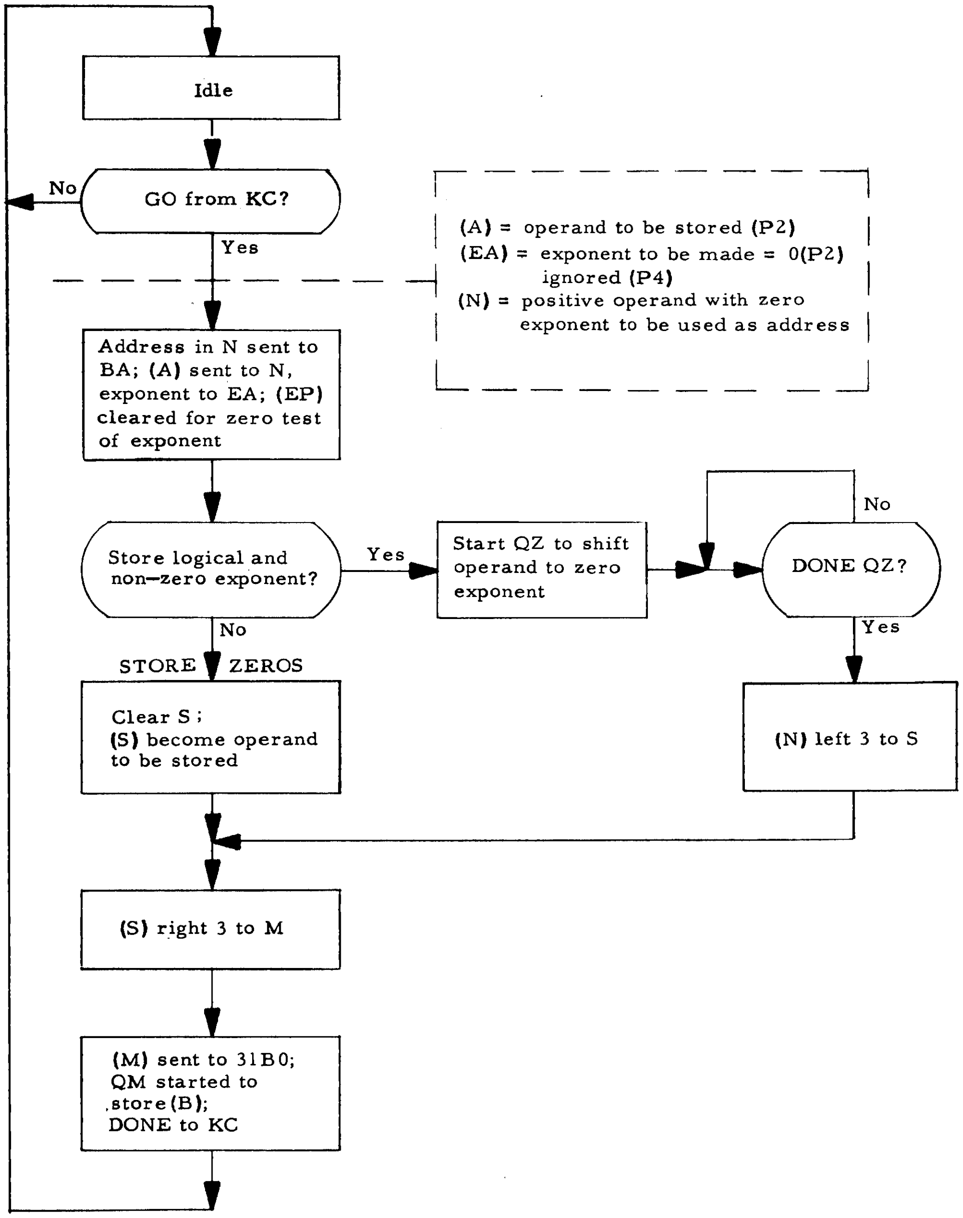
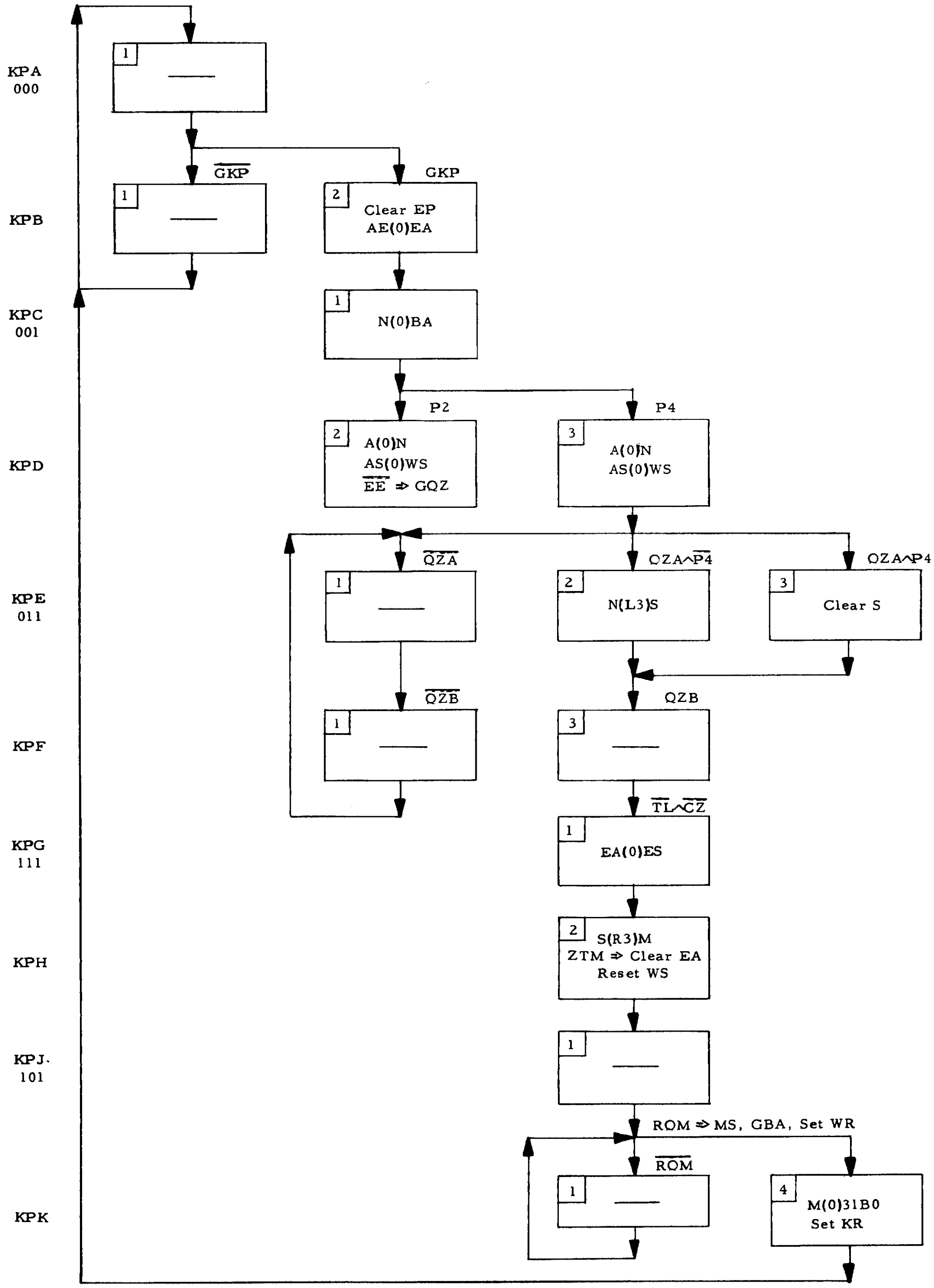

Idle
No
GO from KC?
Yes
(A) = operand to be stored (P2)
(EA) = exponent to be made = 0(P2) ignored (P4)
(N) = positive operand with zero exponent to be used as address
Address in N sent to BA; (A) sent to N, exponent to EA; (EP) cleared for zero test of exponent
Store logical and non–zero exponent?
Yes
Start QZ to shift operand to zero exponent
No
DONE QZ?
Yes
No
STORE ZEROS
Clear S ; (S) become operand to be stored
(N) left 3 to S
(S) right 3 to M
(M) sent to 31B0; QM started to store(B); DONE to KC

TABLE 11.3–2 Terms Used on the KP Flow Chart for the Store Logical and Store Zeros Commands

AS	Accumulator Sign flip–flop; AS is set when the sign of the operand in the Accumulator is negative.
EE	Exponents Equal; (EA) = (EP).
GKP	GO signal to KP sequencer sent by KC for the processing of STORE commands.
GQZ	GO to QZ sequencer to shift the operand to zero or pickapoint exponent.
KR	Opcode DONE signal to KC.
P0, P1, P2, P3, P4	Store commands STD, STS, STL, STI, and STZ respectively.
RQM	READY QM sequencer.
WS	Working Sign flip–flop.
ZTM	Zeros To Memory signal; when high, ZTM indicates that zeros exist in the least significant seven (for non–pickapoint mode) or nine (for pickapoint mode) octals of the operand that is to be stored in the memory.

FIGURE 11.3–4 The KP Flow Chart for the Processing of Store Logical and Store Zero Opcodes

KP



P0 (STD) – store double precision – provides for the floating point storage of the 42 bits of information held in the Accumulator register. This requires the use of the two 32–bit words, locations X and X + 1.

First word: 20(A)0 → 20(X)0, 6(EA)0 → 27(X)21,
sign in WS flip–flop → (X)28,
double precision flag → (X)29,
0 → 31(X)30

Second word: 41(A)21 → 20(X + 1)0,
31(X)21 will be the same as those stored for the first word; on access, these bits are ignored.

KP sends the information required for the first word to the B register and starts QM. QM stores the word in the address specified by (BA), increments (BA) by 1, and sends a READY signal to KP. KP then sends the information from 41(A)21 to the least significant end of the B register, leaving undisturbed the remaining bits in B. QM is started again and, while QM completes the store, KP signals KC of completion of the operation and returns to idle.

When the first word of a double precision number is accessed by KC, the flag in bit 29 is detected and the second half of the number is accessed immediately. This will occur only during the operand assembly operation of KC (for both repeat and non–repeat operations). None of the other access operations call for this interpretation of the information read out.

1) KC, on command access, will interpret the word as a command; thus, the 29th bit will represent mode information;

2) QC is concerned only with command access;

3) KW accesses words for purposes of transmission, not interpretation. It will access and transmit the number of words specified by the block length regardless of the precision and will never investigate the status of the 29th bit.

An interesting implication of the double precision store is that the state of the pickapoint enable does not affect this operation. This makes it possible to perform lengthy pickapoint operations and still insert some information in floating point by calling for double precision storage without necessitating the turning on and off of the pickapoint enable.

P3 (STI) – store integer – provides for storage of the least significant 7 digits of the Accumulator along with the sign, or, if the pickapoint mode is enabled, for storage of the least significant 9 digits of the Accumulator and the sign. In either case, the value is shifted to zero exponent prior to being stored. Lost bits at either end are ignored during the shifting.

1) Floating point integer (pickapoint not enabled):
20(A)0 → 20(X)0
sign in WS → (X)28,
0 → remaining bits

2) Pickapoint integer:
26(A)0 → 26(X)0,
sign in WS → (X)28,
0 → remaining bits.

Note that pickapoint integers are stored with $\overline{\text{B27}}$, pickapoint single precision numbers with B27.

The algorithm and flow chart for these commands are contained in Figures 11.3–5 and 11.3–6.

P1 (STS) — store single precision — provides for floating point storage of the most significant 7 digits of the value held in the Accumulator, or for the pickapoint storage of the most significant 9 digits of this value.

1) Floating point single precision (pickapoint not enabled):
If 41(A)21 = 0 (the operand is not longer than 7 digits)
20(A)0 → 20(X)0. Otherwise, (A) are shifted right, with exponent increments, until 41(A)21 = 0.
6(EA)0 → 27(X)21,
sign in WS flip–flop → (X)28,
0 → 31(X)29
An interrupt is requested if the exponent overflows due to the shifts.

2) Pickapoint single precision (pickapoint enabled):
(A) are shifted left or right, whichever is necessary, to bring the exponent into agreement with the pickapoint exponent. If left shifts generate too large a mantissa, an interrupt is requested; if significant bits are lost due to right shifts, rounding takes place.

   Figures 11.3–7 and 11.3–8 show how these actions are implemented.

Thus, three of the store commands can cause KP to start the QZ sequencer. For the store integer and store logical commands, QZ is always started if the exponent of the operand proves to be other than zero. The exponent checking is handled as follows: the EP register is cleared to zero, and the exponent of the operand to be stored is sent to EA at state KPB. Thus, the comparator circuitry has time, by KPD, to produce EE (exponents equal) if (EA) = (EP) or zero. In either case, if the signal is $\overline{\text{EE}}$, QZ will be started. (Recall that the store integer command, with pickapoint enabled, calls for storage of 9 digits shifted

FIGURE 11.3–5 Algorithm for the Processing of Store Double Precision and Store Integer Opcodes by KP

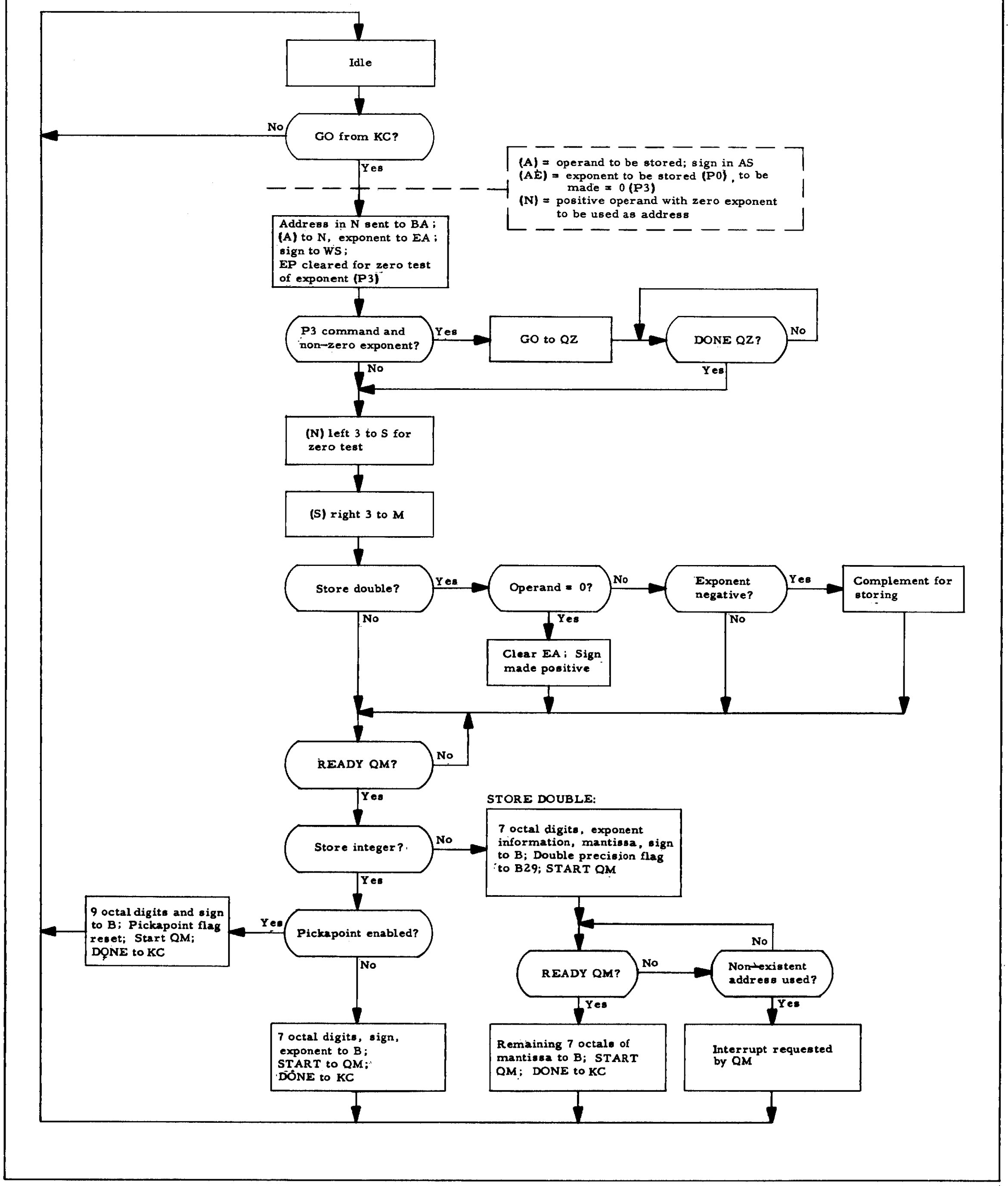

Idle
GO from KC?
No
Yes
(A) = operand to be stored; sign in AS
(AE) = exponent to be stored (P0), to be made = 0 (P3)
(N) = positive operand with zero exponent to be used as address
Address in N sent to BA; (A) to N, exponent to EA; sign to WS; EP cleared for zero test of exponent (P3)
P3 command and non-zero exponent?
Yes
No
GO to QZ
DONE QZ?
No
Yes
(N) left 3 to S for zero test
(S) right 3 to M
Store double?
Yes
No
Operand = 0?
No
Yes
Exponent negative?
Yes
No
Complement for storing
Clear EA; Sign made positive
READY QM?
No
Yes
STORE DOUBLE:
Store integer?
No
Yes
7 octal digits, exponent information, mantissa, sign to B; Double precision flag to B29; START QM
Pickapoint enabled?
Yes
No
9 octal digits and sign to B; Pickapoint flag reset; Start QM; DONE to KC
READY QM?
No
Yes
Non-existent address used?
No
Yes
7 octal digits, sign, exponent to B; START to QM; DONE to KC
Remaining 7 octals of mantissa to B; START QM; DONE to KC
Interrupt requested by QM

TABLE 11.3–3 Terms Used on the KP Flow Chart for Store Double Precision and Store Integer Opcodes

AS	Accumulator Sign flip–flop; AS is set when the sign of the operand in the Accumulator is negative.
Clear EA	Signal that clears exponent register EA to conform to zero operand.
EE	Exponents Equal; (EA) = (EP).
ES7	Bit 7 in register ES; when high, ES7 indicates that the exponent now held in register ES is negative.
GQZ	GO to QZ sequencer to shift the operand to zero or pickapoint exponent.
KR	Opcode DONE signal to KC.
MNA	Memory Non–existent Address; MNA can affect KP only on the STD command since, for the others processed by KP, the KR signal will have been sent to KC and KP will have returned to idle before QM checks the address.
P0, P1, P2, P3, P4	Store commands STD, STS, STL, STI, and STZ, respectively.
QZA, QZB	Idle states of the QZ sequencer; $\overline{QZA}$ indicates that QZ is in operation.
RQM	READY QM sequencer.
TL	Operand Too Large flip–flop; TL is set only for the STS command in non–pickapoint mode; it indicates that the operand is larger than 7 octal digits.
UPE	Pickapoint enable flip–flop; UPE high indicates pickapoint mode.
WS	Working Sign flip–flop.

FIGURE 11.3–6 The KP Flow Chart for the Processing of Store Double Precision and Store Integer Opcodes

KP

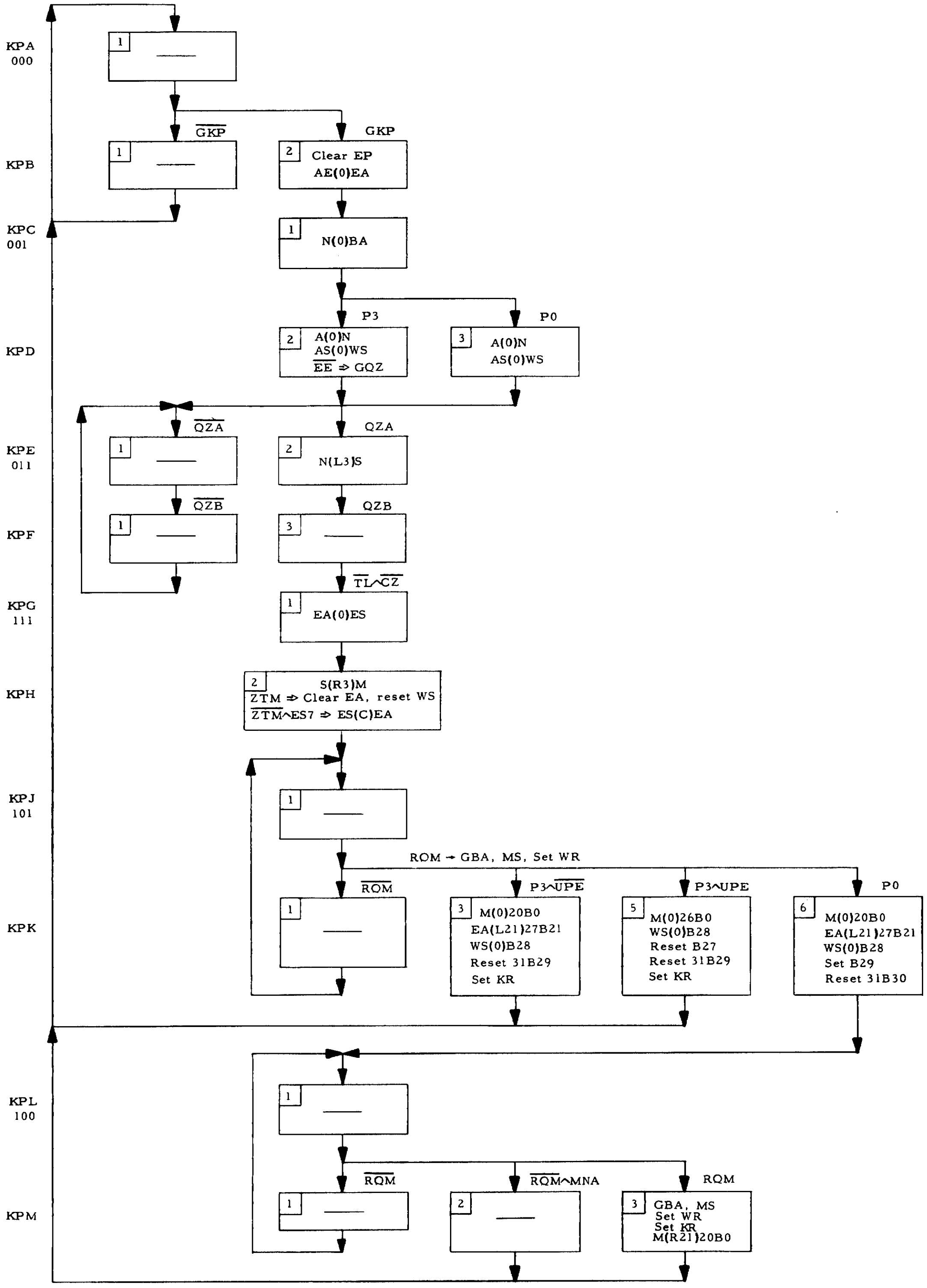


FIGURE 11.3-7 Algorithm for the Processing of Store Single Precision Commands by KP

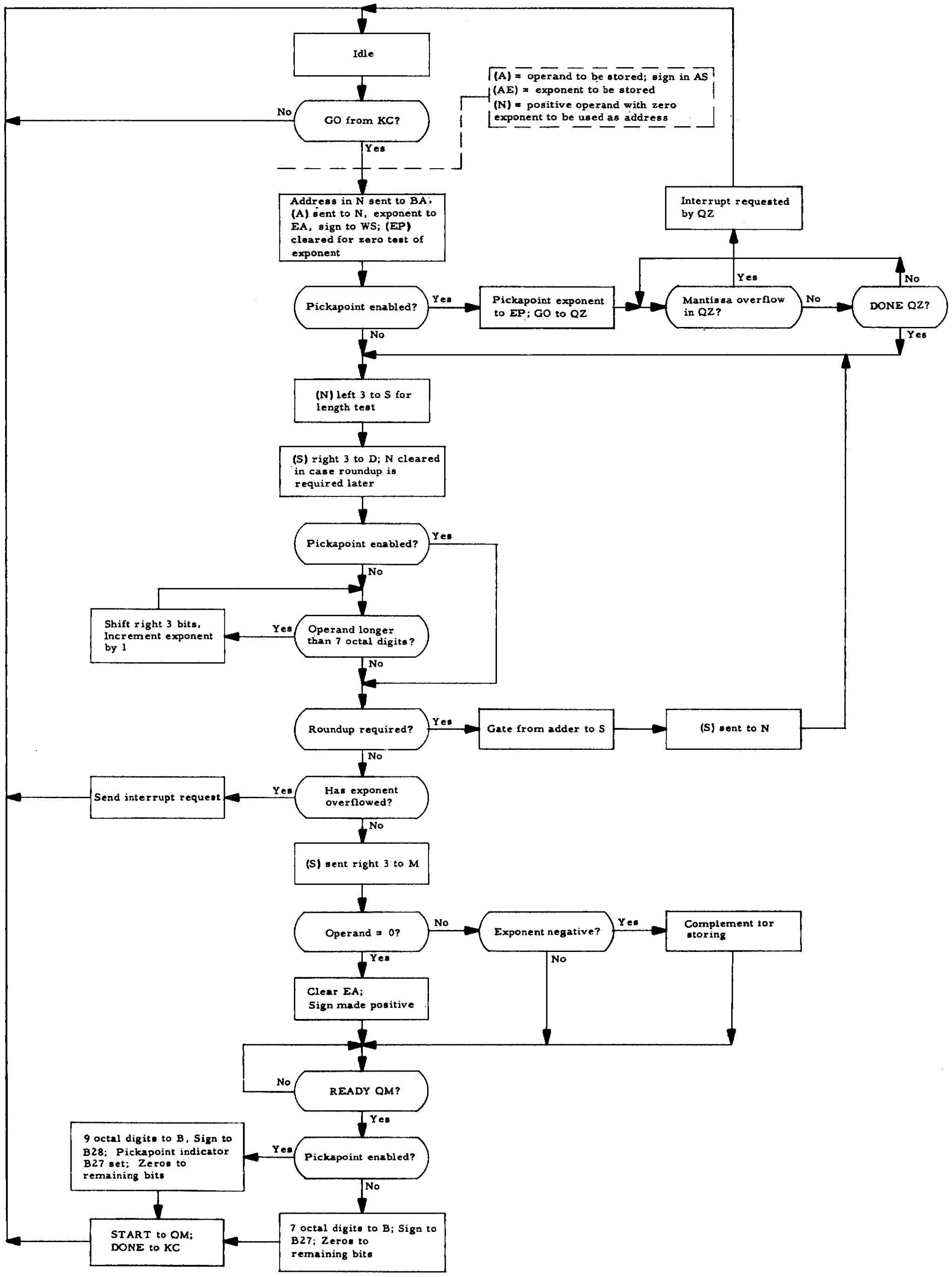
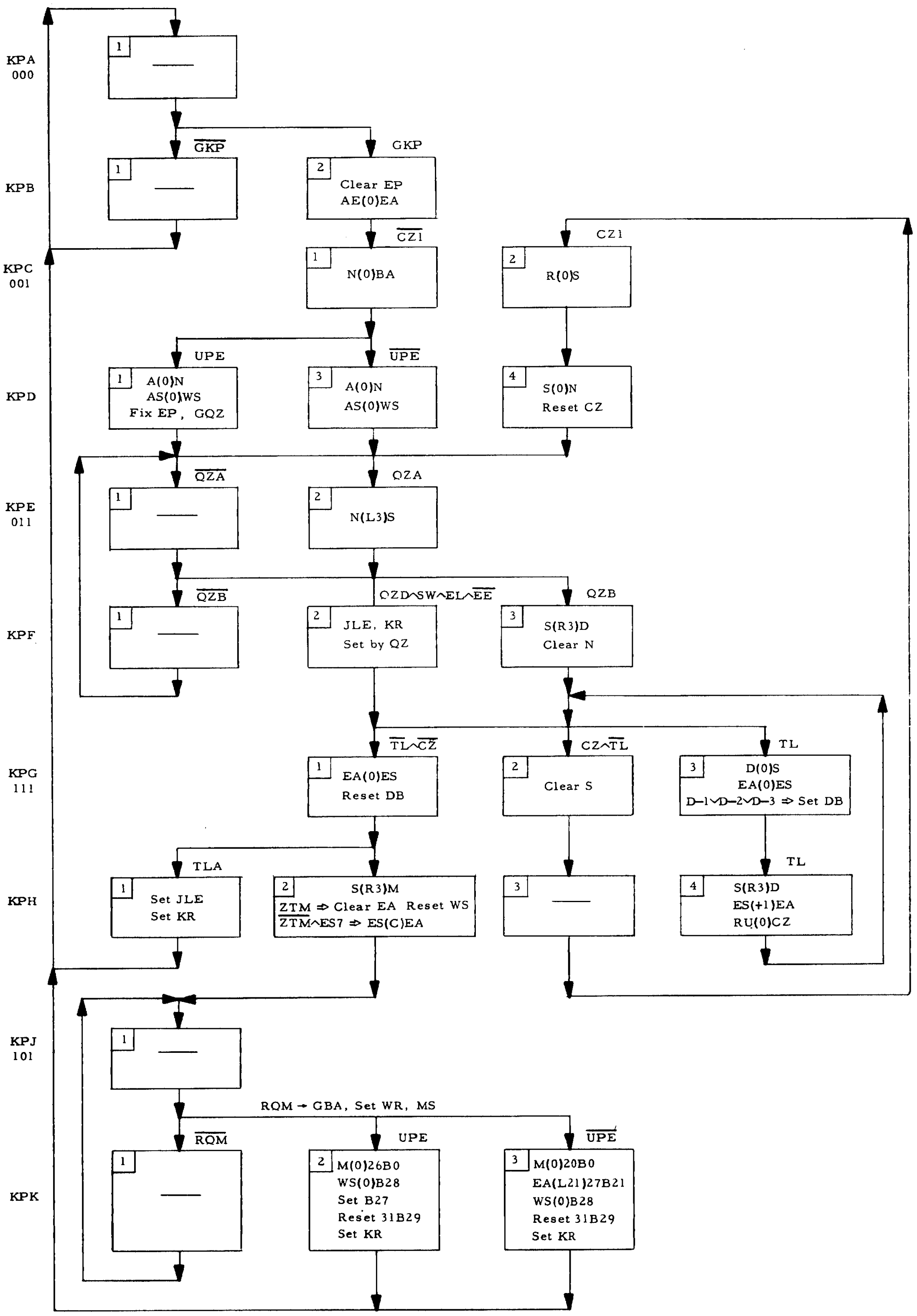

Idle
(A) = operand to be stored; sign in AS
(AE) = exponent to be stored
(N) = positive operand with zero exponent to be used as address
No
GO from KC?
Yes
Address in N sent to BA; (A) sent to N, exponent to EA, sign to WS; (EP) cleared for zero test of exponent
Interrupt requested by QZ
Yes
No
Pickapoint enabled?
Yes
Pickapoint exponent to EP; GO to QZ
Mantissa overflow in QZ?
No
DONE QZ?
No
Yes
(N) left 3 to S for length test
(S) right 3 to D; N cleared in case roundup is required later
Pickapoint enabled?
Yes
No
Shift right 3 bits, Increment exponent by 1
Yes
Operand longer than 7 octal digits?
No
Roundup required?
Yes
Gate from adder to S
(S) sent to N
No
Send interrupt request
Yes
Has exponent overflowed?
No
(S) sent right 3 to M
Operand = 0?
No
Exponent negative?
Yes
Complement for storing
Yes
No
Clear EA; Sign made positive
No
READY QM?
Yes
9 octal digits to B, Sign to B28; Pickapoint indicator B27 set; Zeros to remaining bits
Yes
Pickapoint enabled?
No
START to QM; DONE to KC
7 octal digits to B; Sign to B27; Zeros to remaining bits

TABLE 11.3–4 Terms Used on the KP Flow Chart for Processing Store Single Precision Commands

AS	Accumulator Sign flip–flop; AS is set when the sign of the operand in the Accumulator is negative.
Clear EA	Signal that clears exponent register EA to conform to zero operand.
CZ	Carry–Zero flip–flop; CZ is set to cause a carry–in to the first bit of the Adder. This flip–flop is used by KP only when the STS command is being processed. It can be set by CZ for use by KP when the STS command is written in the pickapoint mode (see Chapter 8), or by KP for the case where the STS command is not written in the pickapoint mode and shifting is required in order to reduce the operand to 7 octal digits. The CZ flip–flop is set by the enabling of the RU(0)CZ path when RU is high.
DB	Discarded Bits flip–flop; DB is set when any of the 3 bits shifted out of the D register on a right shift is a 1; this information is used in establishing the condition of the roundup flip–flop. In KP, this will affect only the shifting necessitated by too large an operand on the STS command; DB $\Rightarrow$ D–1 $\vee$ D–2 $\vee$ D–3.
EE	Exponents Equal; (EA) = (EP).
EL	Exponent Larger; (EA) > (EP).
ES7	Bit 7 in register ES; when high, ES7 indicates that the exponent now held in register ES is negative.
Fix EP	When the STS command is written in the pickapoint mode, the pickapoint exponent is sent to register EP and QZ is started to shift the operand until its exponent value is the same as that in EP or an error occurs. Fix EP refers to the transfer of the pickapoint value from PE to EP. The path enabled is PE(0)EP if the current pickapoint exponent is positive, PE(C)EP if negative.
GKP	GO signal to KP sequencer sent by KC for the processing of STORE commands.
GQZ	GO to QZ sequencer to shift the operand to zero or pickapoint exponent.
JLE	Interrupt request set by KP when shifting is necessitated for the STS command in non–pickapoint mode if the exponent overflows, or by QZ for the STS command in pickapoint mode if the mantissa overflows. (An exponent overflow will not occur in QZ since the pickapoint exponent is known to be in the normal range.)
KR	Opcode DONE signal to KC.
MNA	Memory Non–existent Address; MNA can affect KP only on the STD command since, for the others processed by KP, the KR signal will have been sent to KC and KP will have returned to idle before QM checks the address.
P0, P1, P2, P3, P4	Store commands STD, STS, STL, STI, and STZ, respectively.
RU	Round–up signal; RU is high when rounding is required. The non–biased round–up rule states that round–up takes place if the least significant octal digit (the next one to be discarded by an S(R3)D transfer) is greater than half. If this digit is exactly one–half and the least significant digit retained is even, round–up. Otherwise, truncate. S2 defines one–half since, after the S(R3)D transfer, this will be 2^{-1}. S0 $\vee$ S1 $\vee$ DB indicates greater than one–half if S2 is high, and S3 indicates that the least significant digit retained is even. Thus, RU $\Rightarrow$ S2 $\wedge$ (S0 $\vee$ S1 $\vee$ $\overline{\text{S3}}$ $\vee$ DB).
RQM	READY QM sequencer.
SW	S Wide signal; when high SW indicates that the value in the S register is too large, i. e., significant bits have been shifted into S44 $\vee$ S43 $\vee$ S42. This signal is noticed in QZ only for shifts necessitated by the STS $\wedge$ UPE command and causes an interrupt to be set.
TL	Operand Too Large flip–flop; TL is set only for the STS command in non–pickapoint mode; it indicates that the operand is larger than 7 octal digits.
TLA	Too Large exponent signal; when high, TLA indicates that the exponent has overflowed due to shifts necessitated in order to bring the operand into the 7 octal digit range on STS and UPE command.
UPE	Pickapoint enable flip–flop; UPE high indicates pickapoint mode.
WS	Working Sign flip–flop.
ZTM	Zeros To Memory signal; when high, ZTM indicates that zeros exist in the least significant 7 (for non–pickapoint mode) or 9 (for pickapoint mode) octals of the operand that is to be stored in the memory.

FIGURE 11.3–8 The KP Flow Chart for Store Single Precision Commands

KP



to zero exponent.)

The store single precision command will cause startup of QZ only if the pickapoint mode is enabled. (Otherwise, the value is stored with its own exponent.) In the case of the pickapoint mode, the pickapoint exponent is sent to register EP and the operand is shifted until its exponent agrees with that in EP. Nine rather than 7 digits of the value can be stored since the pickapoint exponent will be held in register PE and need not be kept with each value to which it is appended. An error exit at KPF provides for the case where QZ signals that the left shifts necessitated by the pickapoint mode have caused generation of too large a mantissa. This condition results in an interrupt request. If the shifts to bring about equalization are right shifts, the discarded bits flip–flop will be set as soon as significant bits are shifted out of range and QZ will set CZ so that KP can roundup the value before storing it. KP transfers the operand from N to D, clears N and S, and gates (D) through the Adder with CZ high to obtain the roundup.

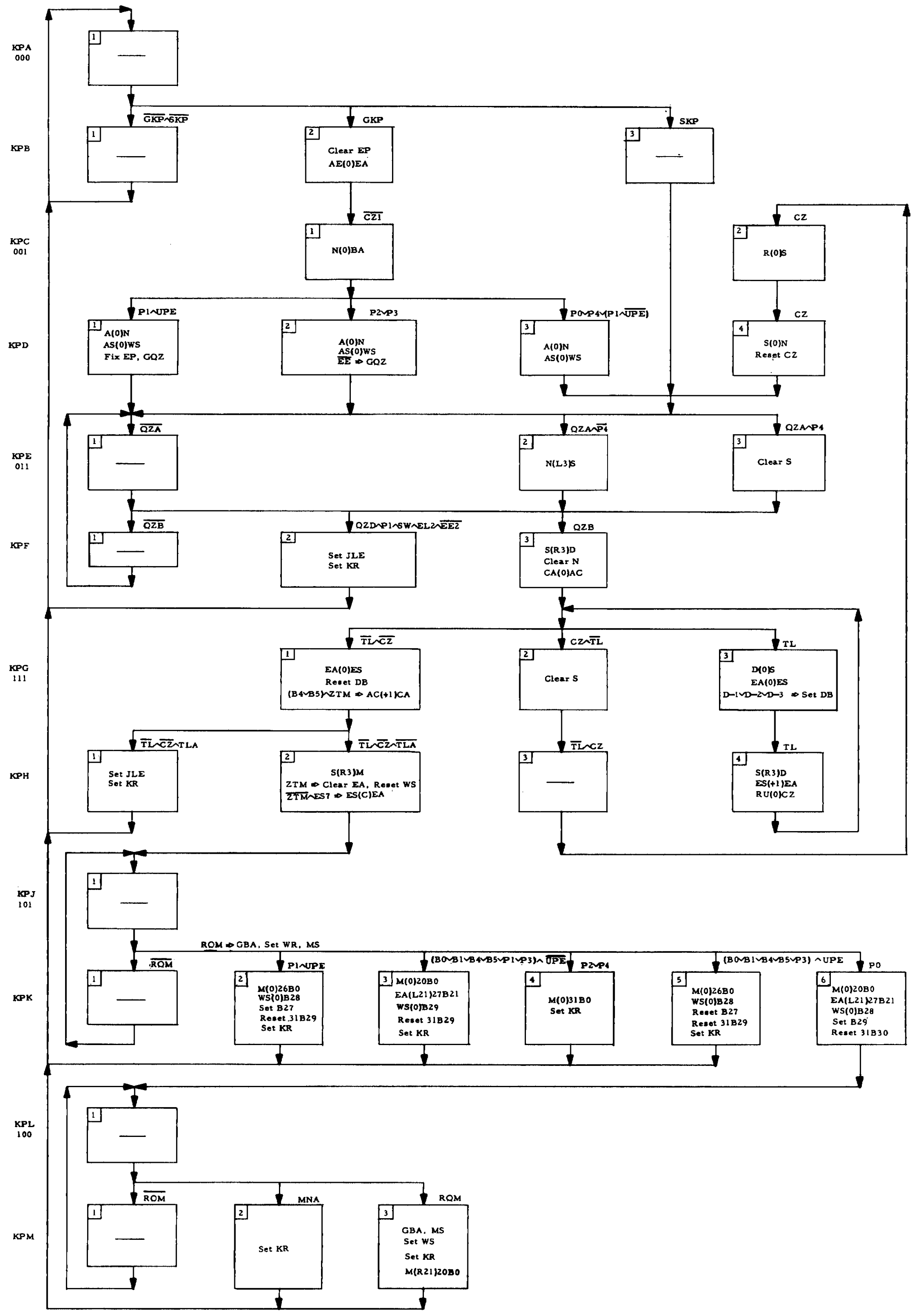
As indicated in the definition of the store single command, the operand will be adjusted so that the most significant bits are within the 20(A)0 range, i.e., operand is no longer than 7 octal digits, before the value is stored. KP tests for this condition by checking 41(A)21 = 0. Actually, at the time the test takes place, the operand is being held in the S register, having been sent there via the left three shift path. Thus, the bits in S which must register zero in order to indicate correct positioning of the value are 44S24. For this reason, the Too Large signal, TL, goes high if 44S24 ≠ 0. With TL high, KP shifts the operand to the right three bits at a time, incrementing the exponent by 1 for each octal shift, until TL goes low, at which point it will roundup the value if CZ has been set during the shift operation.

TABLE 11.3–5 Terms Used on the KP Flow Chart

AS	Accumulator Sign flip–flop; AS is set when the sign of the operand in the Accumulator is negative.
B0, B1, B4, B5	Index opcodes LXP, LXM, XPT, XMT. At opcode startup the decoding of the other four index commands will have been modified to agree with one of these commands so that only four are gated through KP (see Section 9.7).
Clear EA	Signal that clears exponent register EA to conform to zero operand.
CZ	Carry–Zero flip–flop; CZ is set to cause a carry–in to the first bit of the Adder. This flip–flop is used by KP only when the STS command is being processed. It can be set by QZ for use by KP when the STS command is written in the pickapoint mode (see Chapter 8), or by KP for the case where the STS command is not written in the pickapoint mode and shifting is required in order to reduce the operand to 7 octal digits. The CZ flip–flop is set by the enabling of the RU(0)CZ path when RU is high.
DB	Discarded Bits flip–flop; DB is set when any of the 3 bits shifted out of the D register on a right shift is a 1; this information is used in establishing the condition of the round–up flip–flop. In KP, this will affect only the shifting necessitated by too large an operand on the STS command; $DB \Rightarrow D{-}1 \vee D{-}2 \vee D{-}3$.
EE	Exponents Equal; (EA) = (EP).
EL	Exponent Larger; (EA) > (EP).
ES7	Bit 7 in ES register; when high, indicates that the exponent now held in register ES is negative.
Fix EP	When the STS command is written in the pickapoint mode, the pickapoint exponent is sent to register EP and QZ is started to shift the operand until its exponent value is the same as that in EP or an error occurs. Fix EP refers to the transfer of the pickapoint value from PE to EP. The path enabled is PE(0)EP if the current pickapoint exponent is positive, PE(C)EP if negative.
GKP	GO signal to KP sequencer sent by KC for the processing of STORE commands.
GQZ	GO to QZ sequencer to shift the operand to zero or pickapoint exponent.
JLE	Interrupt request set by KP when shifting is necessitated for the STS command in non–pickapoint mode if the exponent overflows, or by QZ for the STS command in pickapoint mode if the mantissa overflows. (An exponent overflow will not occur in QZ since the pickapoint exponent is known to be in the normal range.)
KR	Opcode DONE signal to KC.
MNA	Memory Non–existent Address; MNA can affect KP only on the STD command since, for the others processed by KP, the KR signal will have been sent to KC and KP will have returned to idle before QM checks the address.
P0, P1, P2, P3, P4	Store commands STD, STS, STL, STI, and STZ, respectively.
QZA, QZB	Idle states of the QZ sequencer; $\overline{QZA}$ indicates the QZ is in operation.
QZD	State of the QZ sequencer at which SW is sensed.
RU	Round–up signal; RU is high when rounding is required. The non–biased round–up rule states that round–up takes place if the least significant octal digit (the next one to be discarded by an S(R3)D transfer) is greater than half. If this digit is exactly one–half and the least significant digit retained is even, round–up. Otherwise, truncate. S2 defines one–half since, after the S(R3)D transfer, this will be 2^{-1}. $S0 \vee D1 \vee DB$ indicates greater than one–half if S2 is high, and S3 indicates that the least significant digit retained is even. Thus, $RU \Rightarrow S2 \wedge (S0 \vee S1 \vee \overline{S3} \vee DB)$.
RQM	READY QM sequencer.
SKP	START signal to the KP sequencer for index opcodes.
SW	S Wide signal; when high, it indicates that the value in the S register is too large, i.e., significant bits have been shifted into $S44 \vee S43 \vee S42$. This signal is noticed in QZ only for shifts necessitated by the $STS \wedge UPE$ command and causes an interrupt to be set.
TL	Operand Too Large flip–flop; TL is set only for the STS command in non–pickapoint mode; it indicates that the operand is larger than 7 octal digits.
TLA	Too Large exponent signal; when high, TLA indicates that the exponent has overflown due to shifts necessitated in order to bring the operand into the 7 octal digit range on STS and $\overline{UPE}$ command.
UPE	Pickapoint enable flip–flop; UPE high indicates pickapoint mode.
WS	Working Sign flip–flop.
ZTM	Zeros To Memory signal; when high, ZTM indicates that zeros exist in the least significant 7 (for non–pickapoint mode) or 9 (for pickapoint mode) octals of the operand that is to be stored in the memory.

FIGURE 11.3–9 The KP Flow Chart

KP



CHAPTER 12

INPUT/OUTPUT SEQUENCERS

SECTION 12.1 – INPUT/OUTPUT CONTROL

Central Processor input/output logic provides communication between the Central Processor and associated memory system and the input/output equipment which it controls. All input/output operations in the G–20 System are carried out by means of transmission of single characters over the communication system. Peripheral units regard these characters as separate entities; the Central Processor regards them as elements of 32–bit words. Thus, when the Central Processor is transmitting one or more words from its memory, it accesses each in turn and transmits it one character at a time. When it is the receiver, it packs the characters into 32–bit words and stores the words in successive locations in the memory. In this way all data is received by the Central Processor in the form of logic words. Each 32–bit word can be considered as consisting of five 6–bit or four 8–bit characters. The programmer has the option of coding in 6– or 8–bit mode. The packing and unpacking of G–20 words, and the transmission and receipt of the characters, is referred to as block input/output.

When a Central Processor participates in an input/output operation, it is the controlling element whether it assumes the role of receiver or transmitter. That is, it will always be responsible for setting up the operation and for instructing the other device. Peripheral devices are not synchronized with the Central Processor and memory system clock and, in fact, each type of device operates at its own rate of speed. Magnetic tapes, for example, have much faster transfer rates than do card punches. Since the speeds of these external devices are fixed, it must be possible for the Central Processor to keep pace with any of them if information is not to be lost during communication. This makes mandatory an adaptable input/output control. To accomplish this, input/output logic is designed so that it is dependent upon a response being received to each transmission. Manipulations necessary between transmission or receipt of characters by the Central Processor are completed in time for the fastest response to be received. The Central Processor then idles. When the response is received and has been synchronized with the internal clock (when the response can be read) the synchronizing signal, SY, goes high and takes the Central Processor out of its idle loop. The response that sets SY will fall into one of the following categories:

1) a green response (GRN = 002) sent as a code on character lines in answer to an instruction to indicate that everything is normal and the program should not branch;

2) an REQ signal (REQuest for character) received on the separate REQ line in the communication cable as the acknowledgement to receipt of an expected character and request for the next;

3) a character correctly received in response to an REQ signal sent out by the Central Processor;

4) occurrence of an error condition (parity error, incorrect response, etc.) causing the ER signal to come high;

5) any unexpected response. For example, a red response (RED = 003) is sent by some units when a program branch is called for. This is a line command code, but is not decoded

as such. Rather, it causes ER to go high because it does not conform to the expected response. Similarly, receipt of an unexpected END code causes ER to go high.

If ER goes high during synchronization, a transfer will occur in the program when SY becomes available. The command that causes the transfer is that following the input/output command. The occurrence of signal ER causes termination of a block input/output operation regardless of whether or not the end of the block has been reached. Thus, $\mathrm{ER} \wedge \mathrm{SY}$ terminate a block operation and cause the next command to be executed. If ER is not high, and the end of the block has been reached, computation resumes with the next command plus one.

Block input/output operations handle two distinct functions: the instructing and querying of the input/output units, and the performance of data transfers either from a block of words in memory to the instructed device or from that unit to a block of locations in the memory. The Central Processor uses queries and instructions, referred to as line commands, to control input/output operations. Block transmission of line commands always occurs from the Central Processor to the other device. However, the units do use a few line commands as responses. The only one so used that is decoded and understood by the Central Processor is the green code (GRN = 002), the response used by the unit being instructed to indicate that conditions are normal and that it is ready to receive the next line command. If the unit is not ready to proceed, a RED response is sent. The RED code (003) is not decoded, but will cause the ER signal to come high because it is not the expected response, (GRN), and will, thus, cause the block operation to terminate and the program to branch to an error routine.

Two line commands, ERR (005) and END (004), are sometimes used as responses by input/output units during data transfers, ERR or END being sent on detection of an error condition, or END at the end of a block operation, but the Central Processor does not interpret these as line commands. The ERR code causes the right result because it is not a correct response (REQ or a data character), and, thus, signal ER comes high. Similarly, an ER signal results from receipt of an unexpected END code. If END is sent at the end of the block operation, it is ignored because the Central Processor is already aware of the end of the operation and is not expecting any response.

Much of the input/output flexibility of the G–20 System is due to the use of line commands. The structure of any given input/output system can be changed – units added or removed – without requiring drastic revisions in the input/output routines that handle the system. Line commands make it possible to ascertain the availability of units at any given time, thus allowing for program runs with input/output operations of varying lengths without necessitating the leaving of enough time for the operation of maximum length in all cases. They also provide the means for maximizing available input/output facilities during execution of a program. This because the input/output program can assign tasks to several units of the same type, query them to determine which finishes first, and assign the task that should be handled next to the first unit available rather than the predetermined unit that might be tied up for some time.

Six opcodes initiate block input/output operations. Two of these call for transmission of a block of line commands: BTC6 and BTC8.

Line commands are usually packed in the 8–bit mode due to the format used for the CAL command and instruction numerics. Sending of the line command CAL amounts to transmission of the particular unit's call number; sending of instruction numerics, numbers necessary to the positioning of certain of the devices, amounts to transmission of the correct series of numbers. Since call numbers are always assigned in the 2nn range, and instruction numerics in the 1nn range, neither of these codes can be packed into the 6–bit mode. However, all other line commands are in the 00–77 range and can, therefore, be written in the 6–bit mode using the BTC6 command. No provision exists for receipt of line commands; information is always received in the form of data characters.

Four commands are used to handle data transfers: BTD6, BTD8, BRD6 and BRD8. Before any such transfer can take place, the participating unit (whether it is to be the receiver or the transmitter) must be instructed by the Central Processor. This means that it must receive a series of line commands and that it must respond correctly to each of them. The line commands will most often be packed into a block and sent out through use of opcode BTC6 or BTC8. (Coding of these commands one per command word, using the Transmit Line Command opcode, is discussed later.)

Thus, one block operation is used to set up another block operation. As the line commands are received by the peripheral device, they are decoded and a response is returned to the Central Processor. The normal response to these commands is GRN; when this is received, the next command in the block will be sent. If any other response is received, or no response is received for 1 second, the transmission will be terminated and computation will continue with the next command in the program (the error branch).

Consider the setting up of a data transfer from an MT–10 to the Central Processor. A possible series of commands to do this would be: a call to the unit (this consists of the call number of the unit), a query to see if it is ready to accept more instructions (Query Ready –

QRD) and, assuming that the tape is positioned at the correct block address, TRA if it is to be the transmitter in the operation. If the responses to these commands are all green, the tape enters the INSTRUCTED state and remains there until the actual transfer is initiated by another block input/output operation (which sends it to the MESSAGE state) or, through some change in plan, it is called again and sent back to the CALLED state.

When a unit has been instructed to receive or transmit data, the actual data transfer can be initiated by use of the BRD and BTD opcodes. Whichever of these opcodes is used, the starting line command that occurs in the second command word must always be SDT (Start Data Transfer = 010) which informs the other device that the operation for which it has been instructed is ready to begin and sends it to the MESSAGE state. From this point on, the response to each received data character (whether transmitted by the Central Processor or the other device) will be a signal on the separate REQ line.

The operation will end when the Central Processor transmits either an END or ERR code depending upon whether or not an error has occurred during the processing. Receipt of either of these causes the other device to leave the MESSAGE state and, thus, makes it available for the next operation. Actions taken when an error is detected by the input/output unit vary. For example, the tape unit sends ERR to the Central Processor on detection of a parity error. The Central Processor does not decode this signal, but interprets it as an incorrect response so that signal ER comes high. The disc, in one mode of operation, does not stop the operation until the end of the current block when a parity error is detected on a received character. Instead, it sets its own error flip—flop and switches the state of the parity bit to allow correct parity for purposes of transmission. At the end of the block, it transmits ERR. On detection of an error condition, some units send END or ERR, some do not respond. Any of these events cause the Central Processor's ER signal to come high. (ER is high if a response is not received within one second.)

In data transfers, each character transmitted is supplied with a flag in the 9th bit or data flag bit. This is necessary because the same facilities are used for the transmission of data and line commands and the input/output devices must be able to distinguish immediately between the two. This flag is supplied by the hardware in the Central Processor during transmit operations, or by the logic in the input/output device during receive. Any code received by an input/output unit that does not have this flag attached, will be interpreted as a line command. If the code does not correspond to one of the commands pertaining to this particular unit, the unit will not respond. The failure to respond will set ER in the Central Processor at the end of one second.

Parity bits are supplied to the characters being transmitted by the Central Processor circuitry. Parity of data characters is checked by some receiving devices and parity of line commands is checked by all except the MT–10 which checks parity only on its CAL command. Unlike the flag bits, the parity bits are recorded with the data characters in some input/output units. On transmission, parity is checked by the device if the parity bits are stored with the data characters. It is always checked by the Central Processor on each character received. Detection of a parity error by the Central Processor will bring ER high and cause a branch in the program. All input/output devices that do not store parity bits must be capable of adding them to the characters on transmission since the parity will be checked by the Central Processor before the words are packed and stored.

The block of words operated upon in a block operation may be any length from one word to the maximum available number of words in memory. The same logical manipulations are involved in the handling of each word. Thus, block input/output operations are processed as repeat operations with the iterations continuing until the limit specified by the programmer has been reached. All repeat operations are specified by two command words. The first command word contains the block designation 033 in the opcode bits for all block input/output operations.

31	29	27	20	14 0
F	M	0011011	I	A

KC accesses this word, decodes 033, assembles the operand which is the address of the starting location of the block, checks it for integer format, and sends a START signal (SKM) to the KM sequencer (a GO signal (GKM) is sent for repeated arithmetic and logic operations) which sets up all repeat operations. There are no programming restrictions on the use of address modifications in writing such a command.

KM starts the QC sequencer to access the second command word. This word contains the following information: the block input/output opcode to be executed, the starting line command to be sent to the peripheral device, and the length of the block to be used in the operation. Note that this command word must be carefully formulated: no address modification is allowed (KC does not participate in this access and QC handles the word in a simple–minded fashion) and the number of bits available for the opcode and the starting instruction are unique. Only five bits of the input/output opcode are preserved for decoding in this second word and, of these, only three are actually used (see Table 12.1–1). In the opcode list, these commands are given as 7–bit opcodes, but the least significant four bits are zeros and, of these, two are omitted from the command word itself. This makes 8 bits (15 → 22) available for the starting line command (the 6 index bits plus the two from the opcode). In addition to accessing the second word, QC sends appropriate information to the various control registers before returning control to KM. Thus, the operation is entirely set up except for the transmission of the starting line command. In the case of block data transfers, whether the Central Processor is transmitting or receiving, this line command is always Start Data Transfer (SDT = 010); in the block transmission of line commands, the starting line command is the first in the block to be transmitted. This will most often be a call to the unit being addressed, e.g., if the commands are directed to an MT–10 unit with the call number 220, the line command CAL, which in this case consists of the number 220, appears in the second command word. This word will then have the form

31	27	22	14 0
0000	11000	10010000	Block Length

Most input/output operations are handled through use of block commands. It is also possible, however, to transmit single characters or commands. No such commands exist for receipt of single characters or commands. Only one word is necessary to stipulate such an operation. The commands are Transmit Line Command (TLC = 157) and Transmit Data Character (TDC = 117). For these commands, the operand assembled by KC is the character or command that will be transmitted. Use of these opcodes to instruct a device, or to effect a data transfer, obviously requires more execution time than use of the block mode since each character or command requires accessing of a separate command word and the opcode must be decoded for each. There are situations, however, in which it is appropriate to send only a single character or a single query to an input/output device so that these commands do have a place in the input/output opcode structure. Since these are not repeat operations, KM is not involved. KC accesses

TABLE 12.1–1 Opcode in Second Command Word of Block Input/Output

COMMAND:	BLOCK TRANSMIT COMMANDS		BLOCK TRANSMIT DATA		BLOCK RECEIVE DATA	
	6	8	6	8	6	8
OCTAL DESIGNATION: = BINARY:	040 0100000	140 1100000	000 0000000	100 1000000	020 0010000	120 1010000
COMMAND AS WRITTEN IN 27 – 23 OF COMMAND WORD	01000	11000	00000	10000	00100	10100
DECODING USED IN CD REGISTER	010	110	000	100	001	101
CD12 (MODE)	0	1	0	1	0	1
CD11 (CLASS OF INFORMATION)	1	1	0	0	0	0
CD10 (DIRECTION OF TRANSFER)	0	0	0	0	1	1

the command word, decodes the 7–bit opcode, and starts KX to handle the transmission. Although 7 bits are used to designate the opcode, consistency has been maintained in the decoding of the most significant three bits. As was the case in the block codes, CD12 = 1 for 8–bit mode (TLC and TDC can be written only in the 8–bit mode), CD11 = 1 for commands, 0 for data, CD10 = 0 for transmit operations.

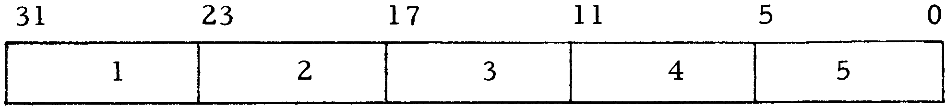
	CD12	CD11	CD10				
TDC	1	0	0	1	1	1	1
TLC	1	1	0	1	1	1	1

Despite the distinctions between the command format and the processing of block and single character input/output operations, the timing considerations and the hardware facilities used for the actual transmission and receipt of information are the same for both. The Line Receiver register (LR) stores incoming information; it consists of 10 bits corresponding to those transmitted over the communication lines: 8 information bits, a data flag, and a parity bit. The Line Driver register (LD) is a pseudo–register (i. e. , the Line Drivers are blocking oscillators, not flip–flops) from which information is transmitted.

Information is sent to the Line Drivers from 33S26 by the enabling of the path S(R26)LD which triggers the Line Drivers. (The data flag and parity bit are supplied by the appropriate circuitry.) Thus, for all transmit operations, the command or character that is to be sent must be shifted into position at 33S26. This means that, at the very least, a left shift of 2 bits is required to position the information since the left–most character of a word is normally positioned at 31S24. In block operations, where successive characters in the word are to be trans–mitted, the word is shifted left between each transmission so that each character is eventually positioned at 33S26 and sent to LD. In single character transmissions, the character must be shifted from the least significant end of the word (where it resides following operand assembly) to this position.

Receive operations also call for left shifts. All information received is packed into 32 bit words, with each character being received in the least significant bits of the S register through enabling of the LR(0)S path. (Parity and flag decoding is handled by the appropriate circuitry.) The word being formed must, then, be shifted 6 or 8 bits (depending upon the mode) to the left between the receipt of each character in order to make room for the next.

In the 6–bit mode, the breakdown of the 32 bits is actually into one 8–bit character followed by four 6–bit characters.



In sending and receiving in this mode, 8 bits will always be transmitted. The most significant character will contain the information from 31S24 of the word, while the most significant two bits of the succeeding characters will be zeroed. If these bits were not zeroed, it would be possible for 1's to be transmitted as part of the character (the least significant two bits of the preceding character are positioned in bits 32 and 33 and the Line Drivers will transmit this information if zeros are not supplied). This would cause the recording of incorrect information in the devices that store all 8 bits on the line. Thus, it is essential that the two leading bits be zeroed for 6–bit mode input/output operations.

In this introductory section, considerable emphasis has been placed on details of input/output because the various operations performed are so interdependent that they cannot be thoroughly understood except in relation to one another. Thus, while transmit differs from receive, and block operations from single transmit operations, there is a basic sameness in the handling of all of them. Most importantly, it should be clear that all input/output operations use the same hardware facilities and that all of them rely on the QW sequencer to coordinate and control these activities. QW is the basic input/output sequencer. It shifts the current word in increments of 6– or 8–bits, sends or receives information by triggering the Line Drivers or sensing the Line Receivers, synchronizes the arrival of the responses, and detects the occurrence of errors. It handles four distinct operations:

1) receive blocks of data words (BRD6, BRD8);

2) transmit blocks of data words or line commands (BTC6, BTC8, BTD6, BTD8);

3) transmit single characters or commands (TLC, TDC);

4) transmit the starting instruction from the second command word of block operations.

In 1) and 2), blocks of information are involved and QW will handle one complete word at a time; in 3) and 4), single characters only are involved; QW will shift this character to 33S26 and transmit it. QW uses the SHS flip–flop to distinguish between block and single character operations. SHS is high for cases 3) and 4) (shifting and transmitting a single character). This tells QW to shift in 6–bit increments until the character is in transmitting position and to trigger the Line Drivers. $\overline{\text{SHS}}$ indicates a block operation. QW uses the CD decoding to gate block operations: CD10 = receive, $\overline{\text{CD10}}$ = transmit, CD12 = shift in 8–bit increments, $\overline{\text{CD12}}$ = shift in 6–bit increments.

The CP–CQ character counter is meaningful in all four cases. This counter, which consists of flip–flops CP1, CP2, CP3, CQ1, CQ2, and CQ3, is designed to keep track of the number of shifts performed. In the case of block operations, this is also the number of characters handled. Counting is done in the Gray code and must go high enough to keep track of five shifts (necessitated by words handled in the 6–bit mode).

TABLE 12.1–2 Counting as Performed by the CP–CQ Counter

Number of Shifts Remaining	CP3	CP2	CP1
4	0	0	0
3	0	0	1
2	0	1	1
1	0	1	0
0	1	1	0

The QW sequencer uses the CP3 signal from this counter. It expects to receive CP3 on the last (or, when SHS is high, the only transmission). Thus, the counter must be initialized so that CP3 will go high correctly. In block operations, the counter is set to 000 when in the 6–bit mode, to 001 for the 8–bit mode. In the case of TLC, TDC commands, the operand to be transmitted must be shifted from 7N0 to 33S26 before transmission. The SHS flip–flop will be set indicating that only one character is to be handled, and the CP–CQ counter will be set to 001. Since SHS shifts occur only in the 6–bit mode, this will send the character 24 bits to the left. The remaining shift (2 bits) will be handled by KX prior to the startup of QW so that, when CP3 comes high, the character will be properly positioned and the transmission may proceed.

In the remaining case, that of transmitting the starting instruction from the second command word of block input/output operations, the character must be shifted from its position in 22S15 to 33S26, or left 11 bits. Since SHS will be high for this operation, shifting will occur in increments of 6 bits. KW, the controlling sequencer for this operation, sets up the 11–bit shift by first shifting the word 1 bit to the right, setting up the CP–CQ counter so that CP3 will come high after two 6–bit shifts have been made, and then starting QW. In this way, when CP3 comes high, the starting command will be positioned at 33S26 for transmission.

This has been a fairly lengthy discussion of factors that are mainly concerned with the QW sequencer. However, these manipulations are basic to all input/output operations and thus, provide necessary orientation to the detailed sequencer descriptions that follow. The remaining sections of this chapter are organized into single character operations and block operations. This means that the KX sequencer, which handles TLC and TDC commands, and the SHS or Shift–Send mode of the QW sequencer are discussed first, followed by the general algorithms for block operations, the KM, QC and KW sequencers, and the $\overline{\text{SHS}}$ mode of QW. Table 12.1–2 is included as a reference for basic input/output information and Table 12.1–3 contains the signals that are used most frequently by all input/output sequencers. It is included at this point to avoid unnecessary redundancy.

One additional topic requires inclusion in this introduction to input/output: that of the initialization or bootstrap mode of operation. This mode of operation is essential since it provides the only method for gaining control over the computer when no control program is in the memory. Bootstrap allows for the input of the control program. In order to use existing logic to perform this operation, key flip–flops are set to artificially advance certain of the sequencers to states they would normally reach midway in an output operation. All other flip–flops are reset, thus effectively clearing the rest of the logic. When the computer is initialized, the initial load switch is thrown. This generates a ZM pulse (Zero Machine) which causes the following:

1) the KM sequencer is advanced to state KMH by means of setting flip–flops KM2 and KM4;

2) the KW sequencer is advanced to state KWL by means of setting flip–flop KW7;

3) all remaining sequencers are cleared to their normal idle loops since the decoding flip–flops are reset;

4) the BS (BootStrap) flip–flop is set to provide special gating in the KM sequencer;

5) flip–flop 7 of the CA register (CA6) is set; this effectively stores address 100_8 (64_{10}) in the address register so that control program will correctly be read in starting at this location;

6) Flip–flops 11 and 13 of the CD register (CD10 and CD12) are set; this establishes the code 101 (receive, 8–bit mode) which is correct for reading in the program;

7) all other flip–flops are reset; since this includes the Master Interrupt Control flip–flop (UJE) no interrupts will be processed during the bootstrap operation; this also clears all registers including register BA which normally holds the block length on an input operation. Thus, no set length is established for the bootstrap program. The device used for bootstrapping is in charge of terminating the operation.

TABLE 12. 1–3 Input/Output Information

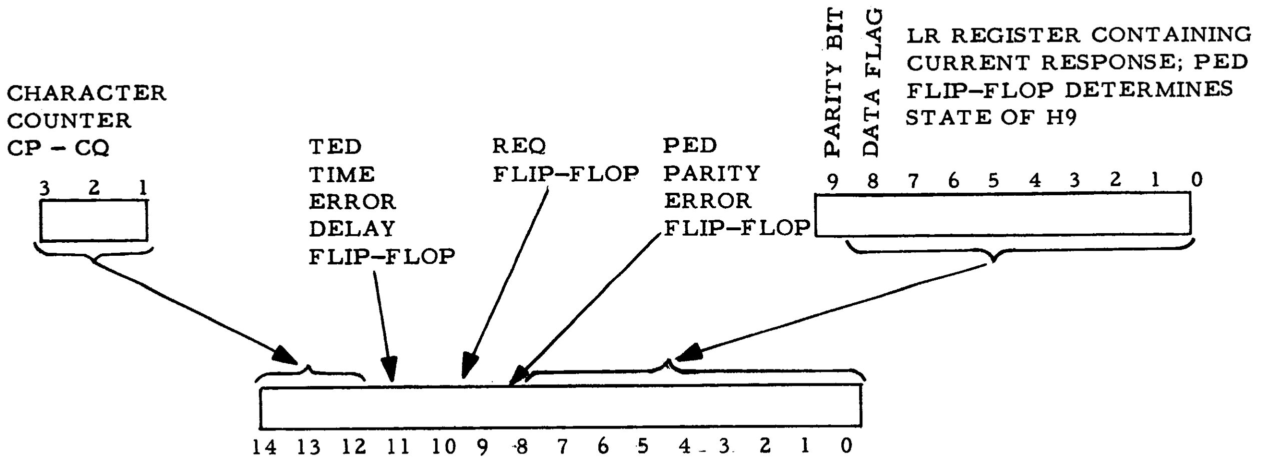
N.B. Only 3 bits are necessary to decode opcode of 2nd word in block command: 27B25 These are sent to CD.	TRANSMIT LINE COMMANDS	TRANSMIT DATA CHARACTERS	BLOCK TRANSMIT COMMANDS		BLOCK TRANSMIT DATA		BLOCK RECEIVE DATA	
			6–BIT	8–BIT	6–BIT	8–BIT	6–BIT	8–BIT
(MODE) CD12	1	1	0	1	0	1	0	1
(CLASS OF INFORMATION) CD11	1	0	1	1	0	0	0	0
(DIRECTION OF TRANSFER) CD10	0	0	0	0	0	0	1	1
	Central Processor is transmitter ——→						receiver ——→	
NORMAL SEQUENCE OF EVENTS: SET UP OPERATION	COMMAND IS TLC OPERAND IS CHARACTER OR COMMAND TO BE SENT	COMMAND IS TDC	TWO COMMAND WORDS: 1st SPECIFIES BLOCK OPERATION (033), OPERAND IS STARTING ADDRESS OF BLOCK; 2nd SPECIFIES OPCODE (SEE DECODING ABOVE), OPERAND IS BLOCK LENGTH, BITS 22–15 CONTAIN STARTING INSTRUCTION					
STARTING INSTRUCTION	———	———	1st COMMAND IN BLOCK		SDT		SDT	
RESPONSE FROM OTHER DEVICE	———	———	GRN	GRN	REQ	REQ	None	
PROCESSOR ACTION:	OPERAND TRANSMITTED		1st OPERAND IN BLOCK IS ACCESSED AND FIRST CHARACTER TRANSMITTED				1st ADDRESS IN BLOCK ACCESSED TO PREPARE FOR STORING OF RECEIVED OPERAND, SEND REQ	
RESPONSE FROM OTHER DEVICE	GRN	GRN	GRN	GRN	REQ	REQ	1st CHARACTER	
PROCESSOR ACTION:	IF NO ERROR CONDITION OCCURS, EXECUTION OF THE PROGRAM WILL CONTINUE AFTER SKIPPING ONE COMMAND		IF NO ERROR CONDITIONS ARISE, CONTINUE OPERATION UNTIL END OF BLOCK IS REACHED; EXECUTION OF THE PROGRAM WILL CONTINUE AFTER ONE COMMAND IS SKIPPED		TRANSMIT END CODE ——→			
ERROR CONDITIONS: POSSIBLE ERRORS	RESPONSE IS NOT GRN CODE ——→ NO RESPONSE RECEIVED FOR 1 SEC. ——→ PARITY ERROR DETECTED ——→		MEMORY OVERFLOW ——→		RESPONSE IS NOT REQ		RESPONSE IS NOT DATA CHARACTER	
ACTIONS TAKEN			TERMINATE (Acc) = ADDRESS OF CURRENT WORD PLUS 2		TRANSMIT ERR CODE; TERMINATE		FILL ANY REMAINING CHARACTER POSITIONS IN CURRENT WORD WITH ZEROS; TRANSMIT ERR CODE; TERMINATE (Acc) = ADDRESS OF CURRENT WORD PLUS 1	

MEMORY OVERFLOW SETS JO FOR INTERRUPT

IN ALL CASES, DETECTION OF AN ERROR CONDITION IS SIGNALED BY ER GOING HIGH; IN BLOCK OPERATIONS, ER CAUSES BRA TO GO HIGH

OCCURRENCE OF AN ERROR CAUSES PROGRAM TO CONTINUE WITH NEXT COMMAND WHICH SHOULD CALL FOR BRANCHING TO AN ERROR ROUTINE

THE PATH LR(0)H IS ENABLED CAUSING THE FOLLOWING INFORMATION TRANSFERS INTO THE H REGISTER


PARITY BIT
DATA FLAG
LR REGISTER CONTAINING CURRENT RESPONSE; PED FLIP-FLOP DETERMINES STATE OF H9
CHARACTER COUNTER CP – CQ
3 2 1
TED TIME ERROR DELAY FLIP-FLOP
REQ FLIP-FLOP
PED PARITY ERROR FLIP-FLOP
9 8 7 6 5 4 3 2 1 0
14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

THUS THE ERROR ROUTINE CAN DETERMINE THE CAUSE OF THE ERROR THROUGH USE OF ERO AND ERA COMMANDS. NOTE THAT IF REQ IS PRESENT AND IS INCORRECT, H10 IS HIGH, 9H0 ARE CLEARED. ALSO, ON BLOCK RECEIVE INCOMPLETE WORDS ARE FILLED WITH ZEROS AND STORED; THUS, THE CP COUNT WILL ALWAYS INDICATE A FULL WORD IN THIS CASE.

TABLE 12. 1–4 Input/Output Information

BA15	Bit 15 of register BA; this flip–flop is set by ZBA high and the enabling of the AC(+1)BA path and indicates that the end of the block has been reached.
BLA	Block Line Amplifiers flip–flop; when high, it inhibits LR inputs; BLA goes low one–half clock time after a transmission if a response is expected.
BRA	BRAnch flip–flop; BRA is set when ER is high; the input/output program will branch on execution of the next command.
CP, CQ	Character counter flip–flops; CP3 is high when shifting is completed for the current operation of QW.
ER	ERror signal; ER high indicates that an incorrect response has been received by the Central Processor or that the response calls for a branch in the program.
PED	Parity Error Detected inverter; PED is high when such an error has been detected in the character received.
RL	Reset LR register flip–flop; when RL is set it causes register LR to be cleared (each flip–flop to be reset) and the BLA flip–flop to be reset (to enable receipt of inputs).
SHS	Shift–Send flip–flop; when set indicates to QW that it is handling a single character, i. e., QW shifts in 6–bit increments until CP3 is high and then transmits the character positioned at 33S26; used in processing TLC and TDC and, in block input/output, to transmit the starting instruction for the second command word.
SY1, SY2, SYA, SYB, SYC, SYD	Synchronize input flip–flops; when high, indicate that response has been received and synchronized with the G–20 clock or that no response has been received for one second.
TSB	Transmit Six Bit mode inverter; $\overline{\text{CD12}}$ causes TSB to go high and remain high after transmission of starting instruction (all single character transmission is in the 8–bit mode). This signal controls the transfer of LR to S; the transfer will contain 6 bits if TSB is high, 8 bits if low.
TDA	Transmit Data inverter; $\overline{\text{CD11}}$ causes TDA to go high; TDA is used to code data flag, LD8.
ZBA	Zero BA signal; ZBA is high when (AC) = 77777_8. ZBA, combined with the enabling of the AC(+1)BA path, causes BA15 to go high.

SECTION 12.2 – THE KX SEQUENCER

The KX sequencer is started by KC to handle processing of the Transmit Single Data Character (TDC) and Transmit Single Line Command (TLC) opcodes. At opcode startup, the assembled operand, in integer format, is positioned in the least significant bits of the N register. Bits 7N0 contain the character or command that will be transmitted. Before the transmission can take place, the operand must be shifted to the sending position in register S: 33S26, or 26 bits to the left. Twenty–four bits of the shift are handled by the QW sequencer. Recall that, when QW operates in the Shift–Send mode, it shifts and transmits only one character with shifts continuing in increments of 6 bits until CP3 signals completion. Thus, KX calls for four such shifts by setting the CP–CQ counter to 001. KX initially shifts the operand 2 bits to the left so that when CP3 comes high, the operand is correctly positioned at 33S26. (KX performs this shift by means of two 1–bit shifts because there is no direct 2–bit–left shift path.) KX must, then, perform the following tasks before starting QW:

1) Set the SHS flip–flop;
2) Clear the Line Response register: Set RL ⇒ clear LR and reset BLA. Following this, BLA is set to inhibit inputs to LR so that the character transmitted will not be heard; QW will enable receipt of the response after the character is transmitted.
3) Reset the SY flip–flop so that synchronization can be signalled;
4) Shift the operand 2 bits to the left and set the CP–CQ counter to 001.

Note that, while shifting with SHS high is always in 6–bit increments, the TLC and TDC commands can be written only in the 8–bit mode, i.e., CD12 is always high for these commands.

When the response from the other device has been received and synchronized, or no response has been received for one second, KX proceeds on the basis of the state of signal ER. If no error has occurred, the address of the next command is incremented by one; if an error has occurred, no increment takes place. (As with all input/output commands, the command immediately following calls for a transfer to the error routine while the normal procedure calls for the skipping of one command.)

FIGURE 12.2–1 Algorithm for the KX Sequencer

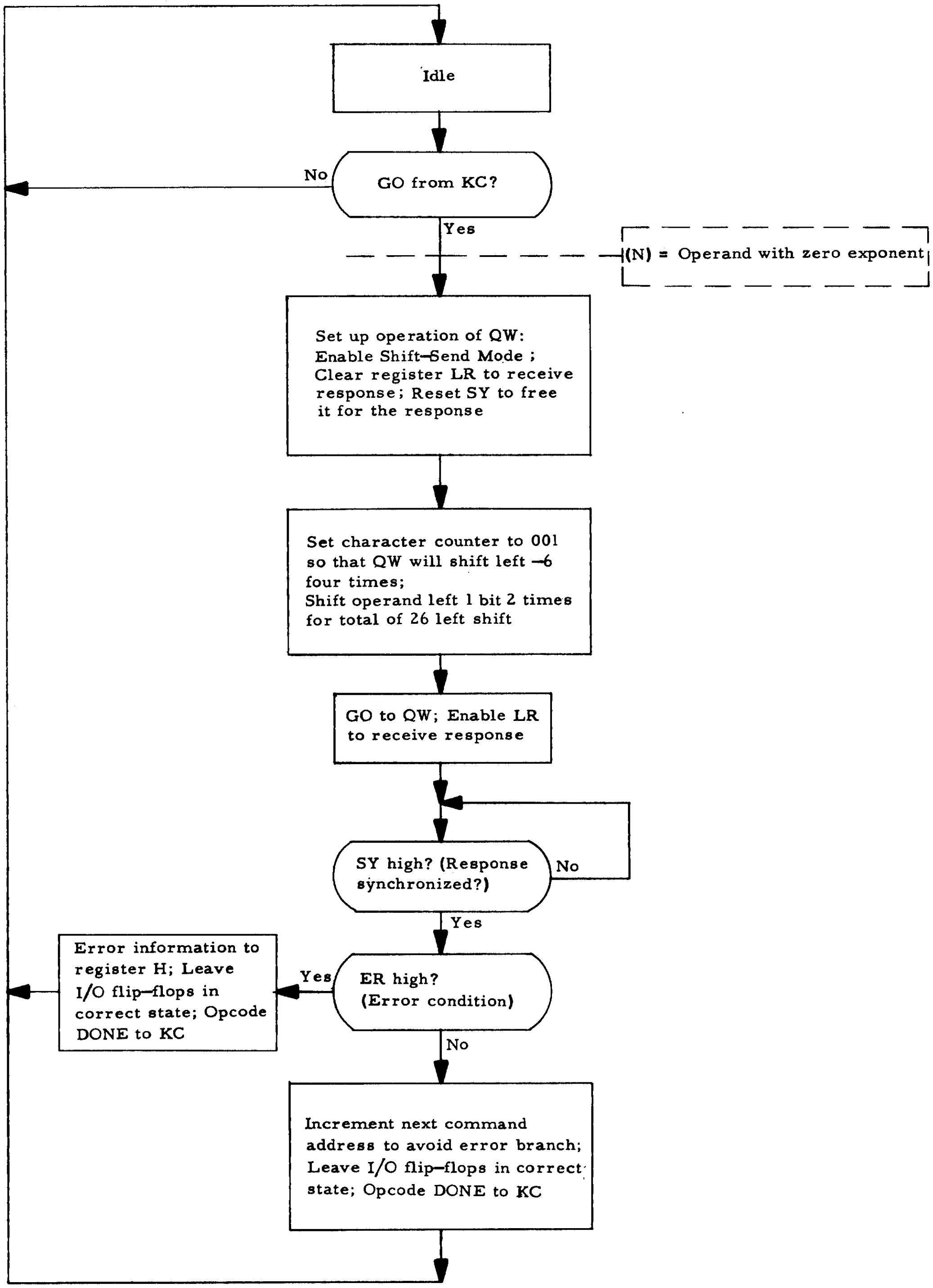
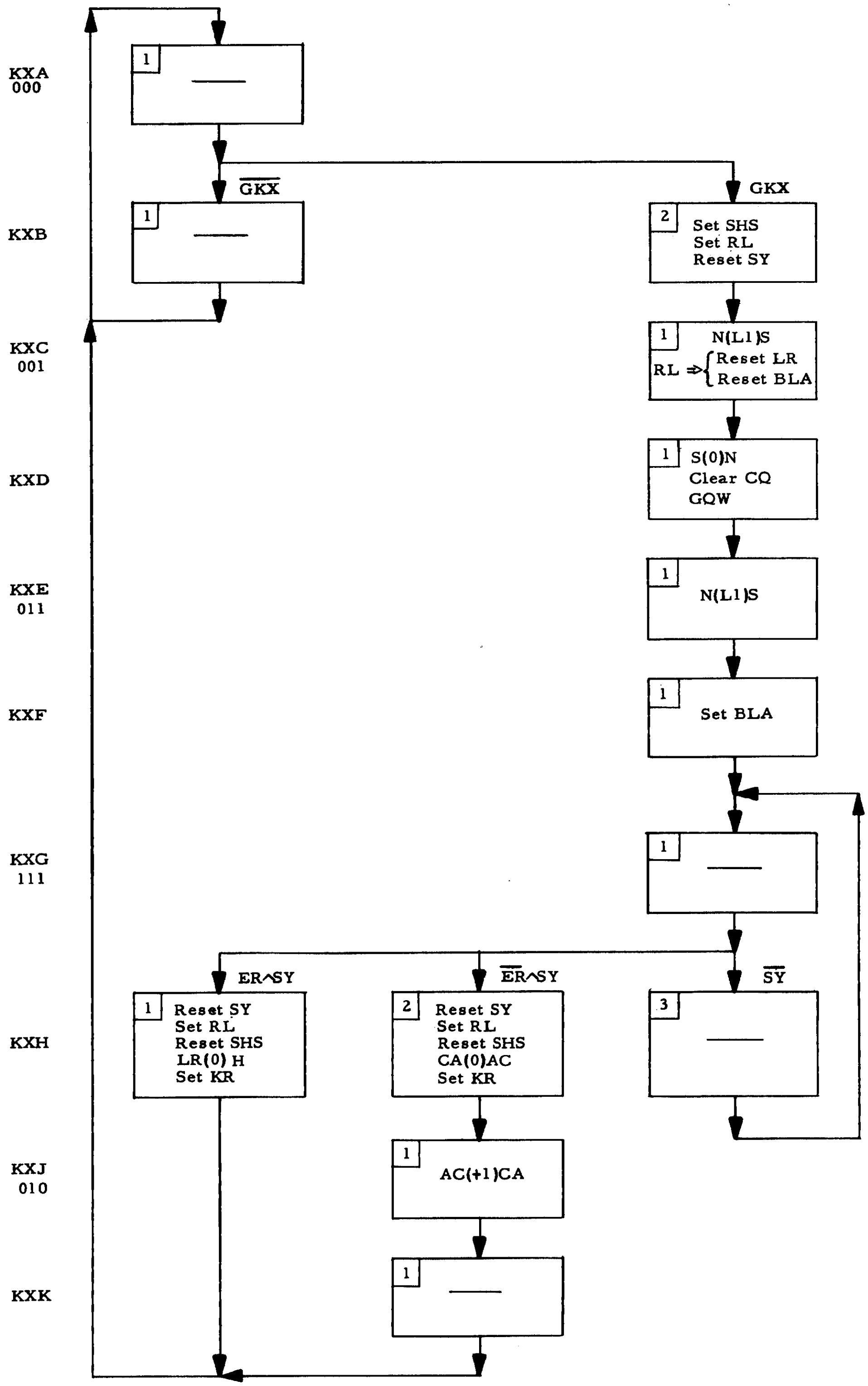

Idle
GO from KC?
No
Yes
(N) = Operand with zero exponent
Set up operation of QW: Enable Shift–Send Mode ; Clear register LR to receive response; Reset SY to free it for the response
Set character counter to 001 so that QW will shift left –6 four times; Shift operand left 1 bit 2 times for total of 26 left shift
GO to QW; Enable LR to receive response
SY high? (Response synchronized?)
No
Yes
ER high? (Error condition)
Yes
Error information to register H; Leave I/O flip–flops in correct state; Opcode DONE to KC
No
Increment next command address to avoid error branch; Leave I/O flip–flops in correct state; Opcode DONE to KC

TABLE 12.2–1 Terms Used in the KX Flow Chart

BLA	Block Line Amplifier flip–flop; reset to enable response to enter register LR input.
Clear CQ	Set up CP–CQ counter to handle appropriate shift; in KX it is set to 001 to call for four shifts.
ER	ERror signal; ER high indicates that an incorrect response has been received by the Central Processor or that the response calls for a branch in the program.
LR(0)H	Path enabled when ER is high to send last response received and error information to register H (see Table 12.1–3).
SHS	Shift–Send flip–flop; SHS high causes QW to shift a single character in 6–bit increments until CP3 is high and then transmit.
SY	SYnchronization flip–flops.
RL	Reset Line Receiver register flip–flop; Set RL ⇒ reset LR register and reset BLA.

FIGURE 12.2-2 The KX Flow Chart

KX



SECTION 12.3 – QW OPERATING IN THE SHIFT–SEND MODE

When QW is started with the SHS flip–flop set, it performs 6–bit shifts until CP3 goes high to indicate that the operand is properly positioned, triggers the Line Drivers, and signals that the operation is complete. This mode of operation is used for transmitting the starting instruction from the second command word in block operations and for transmitting the operand for the TLC and TDC commands. In both instances, the SHS flip–flop must be set, and the CP–CQ counter must be correctly initialized. In the case of the block operation, the KM sequencer (multiple access) sets the SHS flip–flop, while the KW sequencer sets the counter to 011. KW also performs a right–1–bit shift so that the left–12–bit shift called for by this setting of the character counter will result in a total shift of left–11, the correct amount to send the starting instruction from 22S15 to 33S26. In the case of the TLC, TDC commands, a total left shift of 26 bits is required to send the operand from 7N0 to 33S26. To accomplish this, KX sets the SHS flip–flop and shifts the operand 2 bits to the left, then sets the counter to 001 so that QW will perform the remaining 24–bit shift. With the character properly positioned, QW triggers the line drivers and sets the RL flip–flop to clear the Line Receiver register and enable the response. (The BLA flip–flop has been set prior to entry into QW so that the character transmitted will not be sent to LR.)

The algorithm for Shift–Send appears on Figure 12.3–1, and the flow chart in Figure 12.3–2. Note that shifting in this mode is in increments of 6 bits only.

FIGURE 12.3–1 Algorithm for the Shift–Send Mode of QW

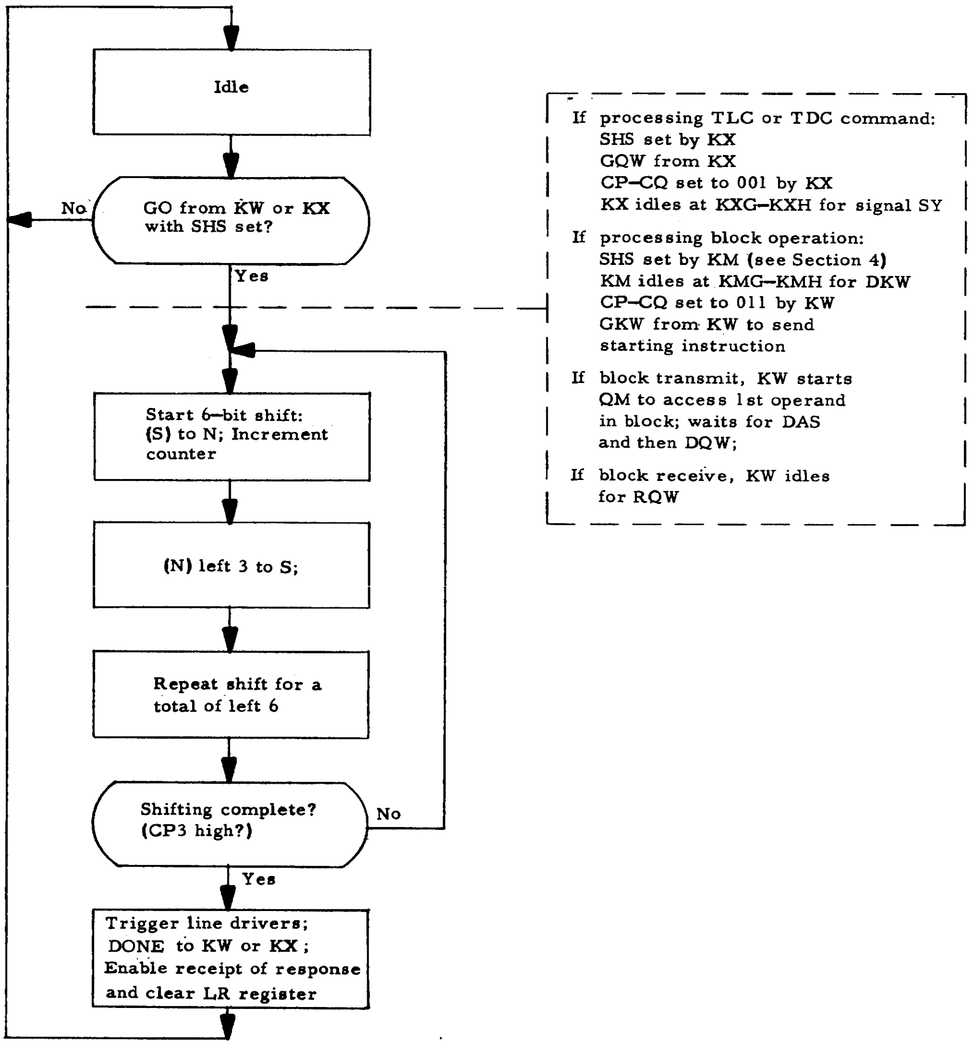
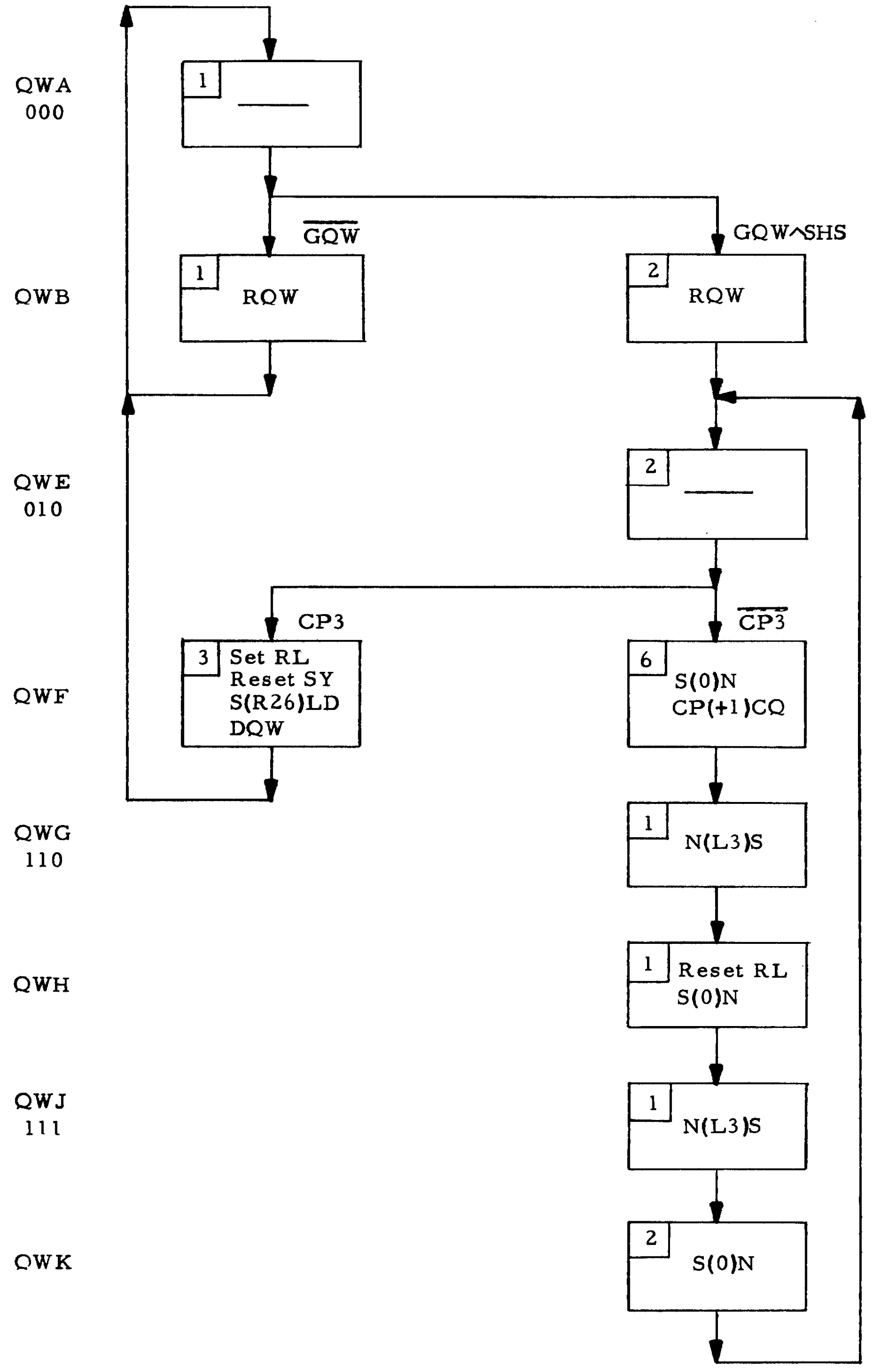


TABLE 12.3–1 Terms Used on the QW Flow Chart for the Shift–Send Mode

CP (+1) CQ	Increment path for CP–CQ counter.
DQW	DONE QW sequencer.
ER	ERror signal; ER high indicates that an incorrect response has been received or that the response calls for a branch in the program.
GQW	GO signal to QW.
LR(0)H	Path enabled when ER is high to send last response received and error information to register H (see Table 12. 1–3).
RL	Reset Line receiver register flip–flop; Set RL ⇒ reset LR, reset BLA.
RQW	READY QW sequencer.
SHS	Shift–Send flip–flop; SHS high causes QW to take paths gated by signal SHS; shift a single character in 6–bit increments until CP3 is high and then transmit.
S(R26)LD	Enabled path that triggers the Line Drivers and causes transmission of the information in 33S26.
SY	SYnchronization flip–flops.

FIGURE 12.3–2 The QW Flow Chart

QW



SECTION 12.4 – BLOCK INPUT/OUTPUT

The general concepts involved in the processing of block input/output commands were discussed in the introduction. The remaining sections in this chapter are devoted to a description of the implementation of this function. The KM sequencer controls block input/output operations, using QC, KW, QW, QM and QA to carry out specific portions of the work. Figure 12.4–1 shows the over–all breakdown of responsibility during block receive (input) operations. KM is in operation from the time of opcode startup until QA is started. (QA sends an opcode DONE signal to KC so that KM need not wait for its completion before returning to idle.)

Figure 12.4–2 shows the path used in implementation of a block receive operation (BRD6 in this case) from the time KM sends a GO signal to KW until the received word has been transferred to the B register so that it is ready to be stored in memory by QM. The enabled paths that affect each transfer are noted, as are the sequencer states at which each occurs. These actions are further related to those taken by KW and QM which proceed with other tasks while QW handles shifting and transmitting of the operand. This example demonstrates the two uses of QW: in the SHS mode for transmission of the starting instruction, and in the normal mode for receipt of data. It should be remembered that the operation being demonstrated would be initiated through use of two command words, and that the second of these would be accessed by the QC sequencer. QC would send the block length to register BA and 5 bits of the opcode to register CD, thus completing the setting up of the operation except for the transmission of the starting instruction. This instruction is held in 22S15 of the second command word so that a left shift of 11 bits is necessary to position it for transmission. Recall that KW shifts this instruction 1 bit to the right before starting QW in the Shift–Send mode, then sets the

CP—CQ counter up to shift left 12 bits, thus effecting a left shift of 11 bits. The paths involved in the shifts are shown in Figure 12.4—2. Following this, the paths used for receipt of characters in the least significant end of the S register and the shifting required to pack the word are demonstrated. The example ends when the word is properly positioned in the B register for storing in memory.

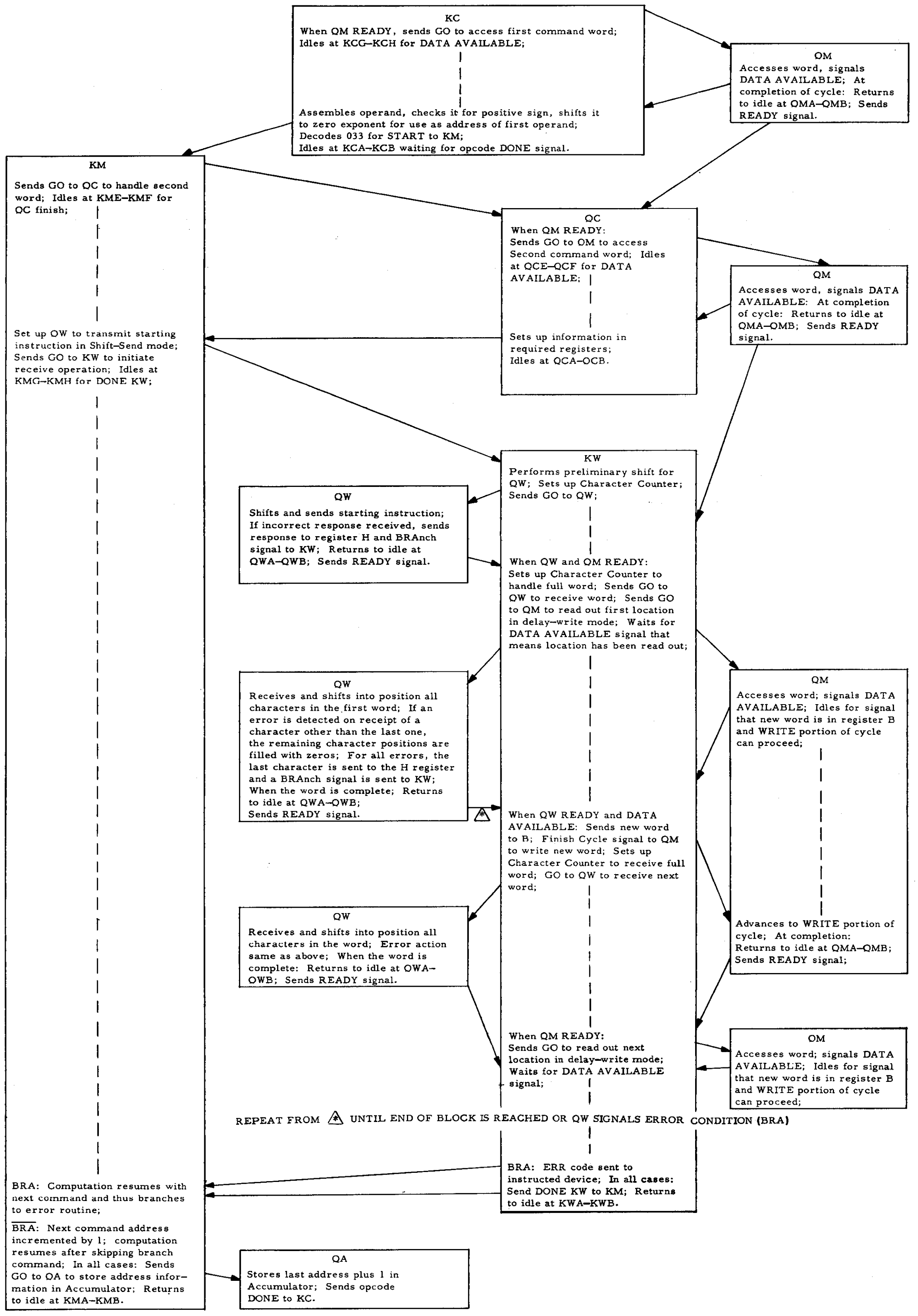
Figure 12.4—3 is a breakdown of the block transmit (output) operation, similar to that given for block receive, while Figure 12.4—4 shows the transfer and shift paths used in implementation of the BTC8 command. Recall that, for the BTC commands, the starting instruction contained in the second command word is actually the first command in the block of commands about to be transmitted. In this example, it is assumed that the first command is a CAL to an MT—10 unit with the call number 220. Thus, the second word has the form

```
 31    27     22          14          0
|0000 |11000 | 10 010 000 |Block Length |
```

This starting instruction will be shifted and transmitted in the same manner as was the SDT in the preceding example. The example follows the action from the time that the first word in the block has been accessed by QM and QW has completed the shifting and sending of the starting instruction to the transmission of the least significant command in the first word.

The remaining sections in the chapter deal with the KM, QC, KW and QW sequencers. Only the block input/output function of KM is included in this chapter; the other half of that sequencer appears in the Adder chapter since it is applicable only to repeat operations. The Shift—Send mode of QW has already been described; hence, the final section describes the operation of QW in the transmission or receipt of full words as it occurs in block operations.

FIGURE 12.4–1 General Algorithm for Block Receive Operations



KC
When QM READY, sends GO to access first command word; Idles at KCG–KCH for DATA AVAILABLE;

Assembles operand, checks it for positive sign, shifts it to zero exponent for use as address of first operand; Decodes 033 for START to KM; Idles at KCA–KCB waiting for opcode DONE signal.

QM
Accesses word, signals DATA AVAILABLE; At completion of cycle: Returns to idle at QMA–QMB; Sends READY signal.

KM
Sends GO to QC to handle second word; Idles at KME–KMF for QC finish;

Set up QW to transmit starting instruction in Shift–Send mode; Sends GO to KW to initiate receive operation; Idles at KMG–KMH for DONE KW;

QC
When QM READY: Sends GO to QM to access Second command word; Idles at QCE–QCF for DATA AVAILABLE;

Sets up information in required registers; Idles at QCA–QCB.

QM
Accesses word, signals DATA AVAILABLE: At completion of cycle: Returns to idle at QMA–QMB; Sends READY signal.

KW
Performs preliminary shift for QW; Sets up Character Counter; Sends GO to QW;

When QW and QM READY: Sets up Character Counter to handle full word; Sends GO to QW to receive word; Sends GO to QM to read out first location in delay–write mode; Waits for DATA AVAILABLE signal that means location has been read out;

When QW READY and DATA AVAILABLE: Sends new word to B; Finish Cycle signal to QM to write new word; Sets up Character Counter to receive full word; GO to QW to receive next word;

When QM READY: Sends GO to read out next location in delay–write mode; Waits for DATA AVAILABLE signal;

QW
Shifts and sends starting instruction; If incorrect response received, sends response to register H and BRAnch signal to KW; Returns to idle at QWA–QWB; Sends READY signal.

QW
Receives and shifts into position all characters in the first word; If an error is detected on receipt of a character other than the last one, the remaining character positions are filled with zeros; For all errors, the last character is sent to the H register and a BRAnch signal is sent to KW; When the word is complete; Returns to idle at QWA–QWB; Sends READY signal.

QM
Accesses word; signals DATA AVAILABLE; Idles for signal that new word is in register B and WRITE portion of cycle can proceed;

Advances to WRITE portion of cycle; At completion: Returns to idle at QMA–QMB; Sends READY signal;

QW
Receives and shifts into position all characters in the word; Error action same as above; When the word is complete: Returns to idle at OWA–OWB; Sends READY signal.

QM
Accesses word; signals DATA AVAILABLE; Idles for signal that new word is in register B and WRITE portion of cycle can proceed;

REPEAT FROM A UNTIL END OF BLOCK IS REACHED OR QW SIGNALS ERROR CONDITION (BRA)

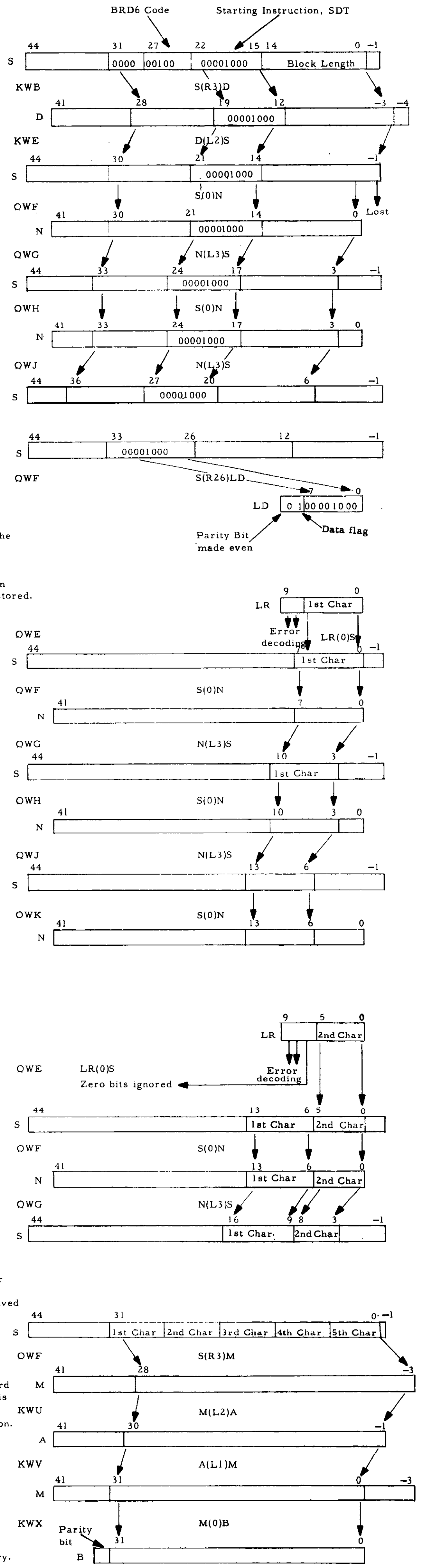
BRA: ERR code sent to instructed device; In all cases: Send DONE KW to KM; Returns to idle at KWA–KWB.

BRA: Computation resumes with next command and thus branches to error routine;

$\overline{\text{BRA}}$: Next command address incremented by 1; computation resumes after skipping branch command; In all cases: Sends GO to QA to store address information in Accumulator; Returns to idle at KMA–KMB.

QA
Stores last address plus 1 in Accumulator; Sends opcode DONE to KC.

FIGURE 12.4–2 Block Receive Example



At GKW:
(S) = 2nd command word minus flags
(CD) = 5 opcode bits
(BA) = 2's complement of block length

Remaining task before block operation begins: t ransmission of starting instruction

Shift left 3 twice more until

No response is expected from the receiver of SDT. KW sets up QW to receive: Sends REQ, fixes counter and starts read out of 1st address in block so that new word can be stored.

QW clears LR; after 1st character, sends out REQ.

If no error and 1st CHAR is synchronized (SY), LR(0)S is enabled

Note that communication lines transmit zeros in bits 7 and 8 for 6–bit mode; these zeros are retained in the most significant character as flag bits to complete 32–bit word.

If an error occurs, incomplete words are filled with zeros; the path LR(0)H is then enabled.

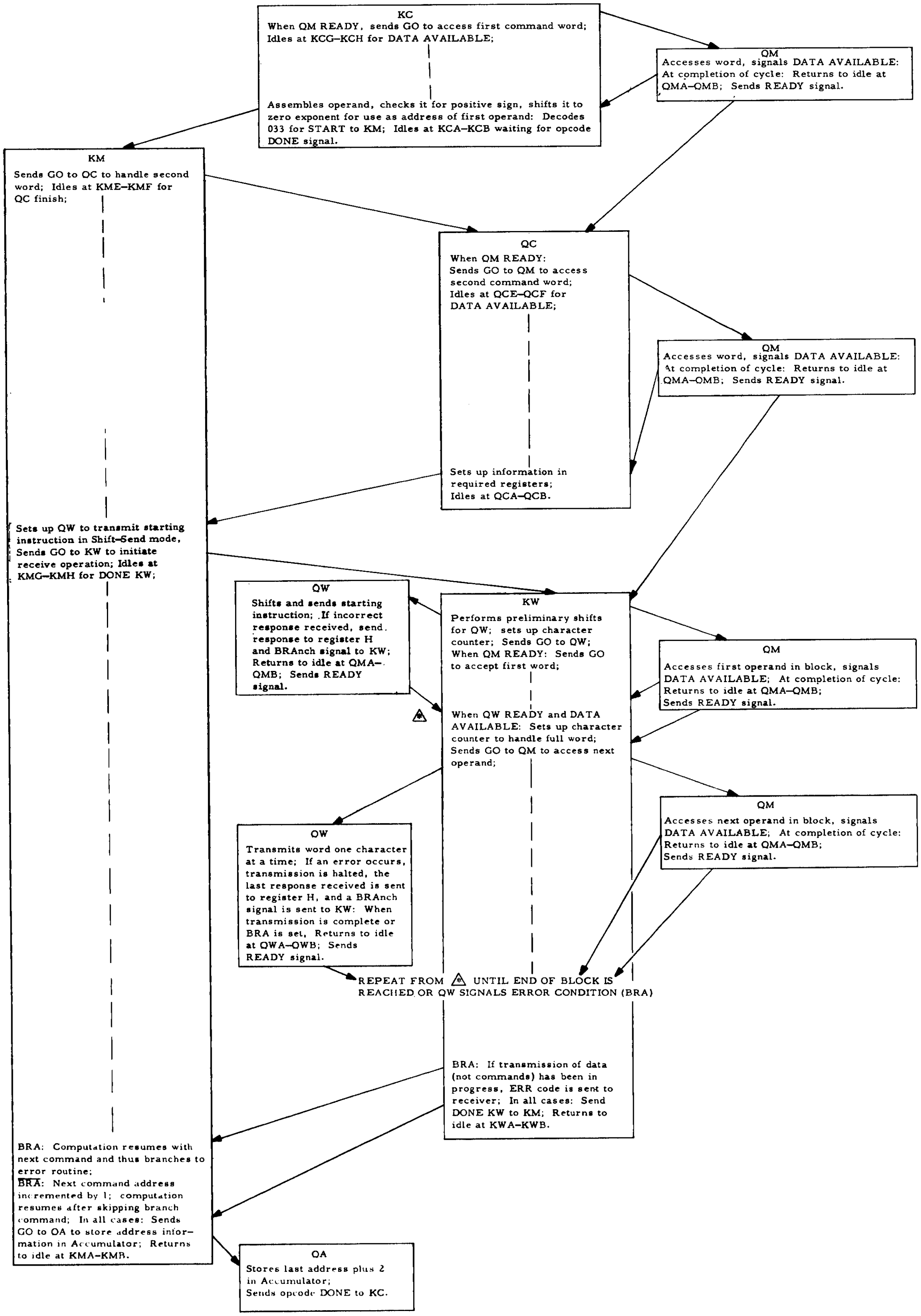
If 2nd character is in LR, no error has occurred, and the character is synchronized, LR(0)S is enabled.

QW continues to receive, error check, synchronize and shift until a full word has been received

QW exits, QM has read out word stored in desired location and is read, to write this one in its place. KW shifts it into position.

From B, it is stored in memory.

FIGURE 12.4–3 General Algorithm for Block Transmit Operations



KC

When QM READY, sends GO to access first command word; Idles at KCG–KCH for DATA AVAILABLE;

QM

Accesses word, signals DATA AVAILABLE: At completion of cycle: Returns to idle at QMA–QMB; Sends READY signal.

Assembles operand, checks it for positive sign, shifts it to zero exponent for use as address of first operand: Decodes 033 for START to KM; Idles at KCA–KCB waiting for opcode DONE signal.

KM

Sends GO to QC to handle second word; Idles at KME–KMF for QC finish;

QC

When QM READY:
Sends GO to QM to access second command word;
Idles at QCE–QCF for DATA AVAILABLE;

QM

Accesses word, signals DATA AVAILABLE: At completion of cycle: Returns to idle at QMA–QMB; Sends READY signal.

Sets up information in required registers;
Idles at QCA–QCB.

Sets up QW to transmit starting instruction in Shift–Send mode, Sends GO to KW to initiate receive operation; Idles at KMG–KMH for DONE KW;

QW

Shifts and sends starting instruction; If incorrect response received, send response to register H and BRAnch signal to KW; Returns to idle at QMA–QMB; Sends READY signal.

KW

Performs preliminary shifts for QW; sets up character counter; Sends GO to QW; When QM READY: Sends GO to accept first word;

QM

Accesses first operand in block, signals DATA AVAILABLE; At completion of cycle: Returns to idle at QMA–QMB;
Sends READY signal.

⚠

When QW READY and DATA AVAILABLE: Sets up character counter to handle full word; Sends GO to QM to access next operand;

QM

Accesses next operand in block, signals DATA AVAILABLE; At completion of cycle: Returns to idle at QMA–QMB;
Sends READY signal.

QW

Transmits word one character at a time; If an error occurs, transmission is halted, the last response received is sent to register H, and a BRAnch signal is sent to KW: When transmission is complete or BRA is set, Returns to idle at QWA–QWB; Sends READY signal.

REPEAT FROM ⚠ UNTIL END OF BLOCK IS REACHED OR QW SIGNALS ERROR CONDITION (BRA)

BRA: If transmission of data (not commands) has been in progress, ERR code is sent to receiver; In all cases: Send DONE KW to KM; Returns to idle at KWA–KWB.

BRA: Computation resumes with next command and thus branches to error routine;
$\overline{\text{BRA}}$: Next command address incremented by 1; computation resumes after skipping branch command; In all cases: Sends GO to OA to store address information in Accumulator; Returns to idle at KMA–KMB.

OA

Stores last address plus 2 in Accumulator;
Sends opcode DONE to KC.

FIGURE 12.4–4 Block Transmit Example

QW transmitted the first command, at the same time KW caused QM to access the first operand in the block. At KWJ–KWK, KW idles until operand is in B and OW is ready to send it

QME block address incremented to 2nd operand

KW starts QW
QW waits for GRN response to set SY ⇒ or for ER to set SY and enable LR(0)H

GRN response to 1st command

LR | 1 | 0 | 00000010 |
⇒ SY

KW starts QM to access 2nd operand; block address incremented to 3rd operand

LR cleared for receipt of next response

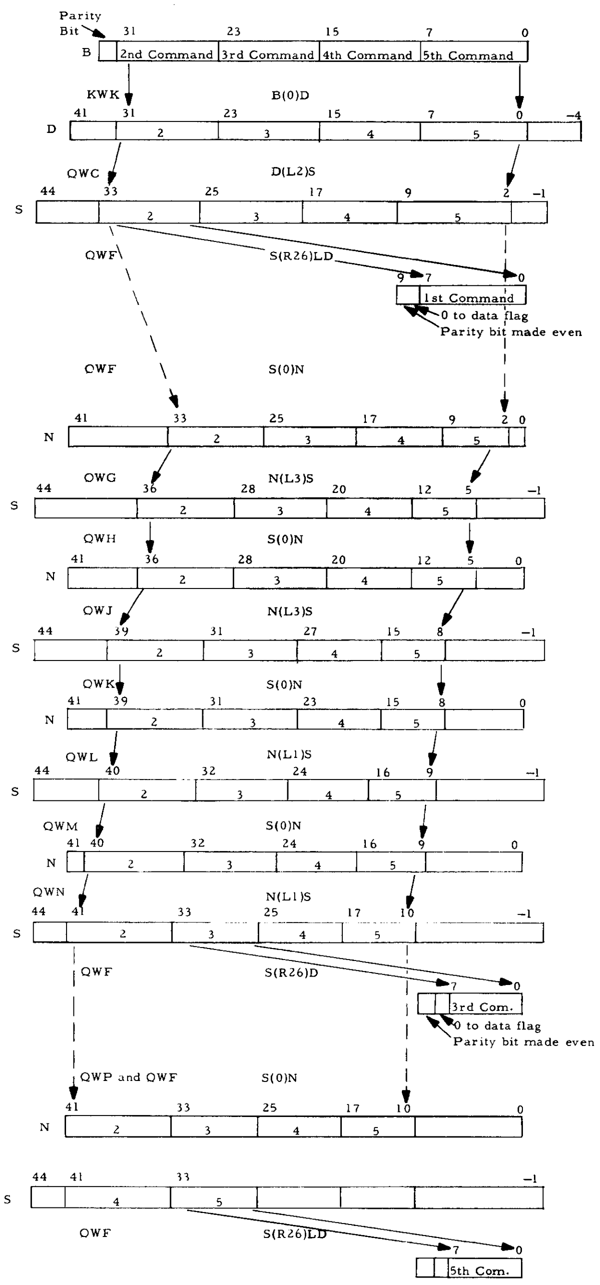
When the second operand is available in B, it is sent to D by KW (KWP)

GRN response to 2nd command

LR | 1 | 0 | 00000010 | (9 0)

No error and SY ⇒ Error and SY enables LR(0)H

OW continues until the last character is sent or an error occurs.



SECTION 12.5 – THE KM SEQUENCER

KM controls all multiple access opcodes. Since block input/output operations involve this type of operation, they are controlled by KM. The charts in Section 4 should provide some orientation to the type of control exercised by this sequencer. It is interesting to note the distinction in methods used to access the operands in the block in the two types of operations controlled by KM. In block input/output, KM uses QM to access the next operand. In repeat arithmetic and logic operations, KC is used to perform the accesses. (This makes it possible to use the KC logic to format the operands correctly and also to connect properly with the sequencer handling the operation.) In repeat operations, all activities are stopped during the access since the operand must be available in order for the arithmetic or logic operation to proceed; in block input/output, operands are handled entirely separately so that, while one is being transmitted, the next can be accessed. In block input/output, these operations are run as closely together as possible with the interesting results shown in the figures in Section 4.

No interrupts are allowed during the processing of a block input/output operation. In fact, there is no way for an interrupt to be processed since the state at which such a decision is made in Master Control, KCA, will not be re–entered until the entire block operation is complete. In order to ensure immediate processing of any enabled interrupts when control returns to KC, the KM sequencer sets the CD8 flip–flop prior to exiting. Recall that interrupts are not processed by KC if the preceding command left useful information in the OA register, i.e., if it were one of the N commands. Only 3 bits are used in the decoding of block input/output commands and, while this is sufficient to distinguish them from one another, it does not distinguish between the block commands and the N commands. Since CD8 is always low for N commands, changing the decoding configuration by means of setting CD8 before returning control to KC allows for the immediate processing of any enabled interrupts. KM also restores the next command address to the CA register before exiting, incrementing it by 1 if nothing has occurred to cause a branch in the program.

FIGURE 12.5–1 Algorithm for the Control of Block Input/Output Operations by KM

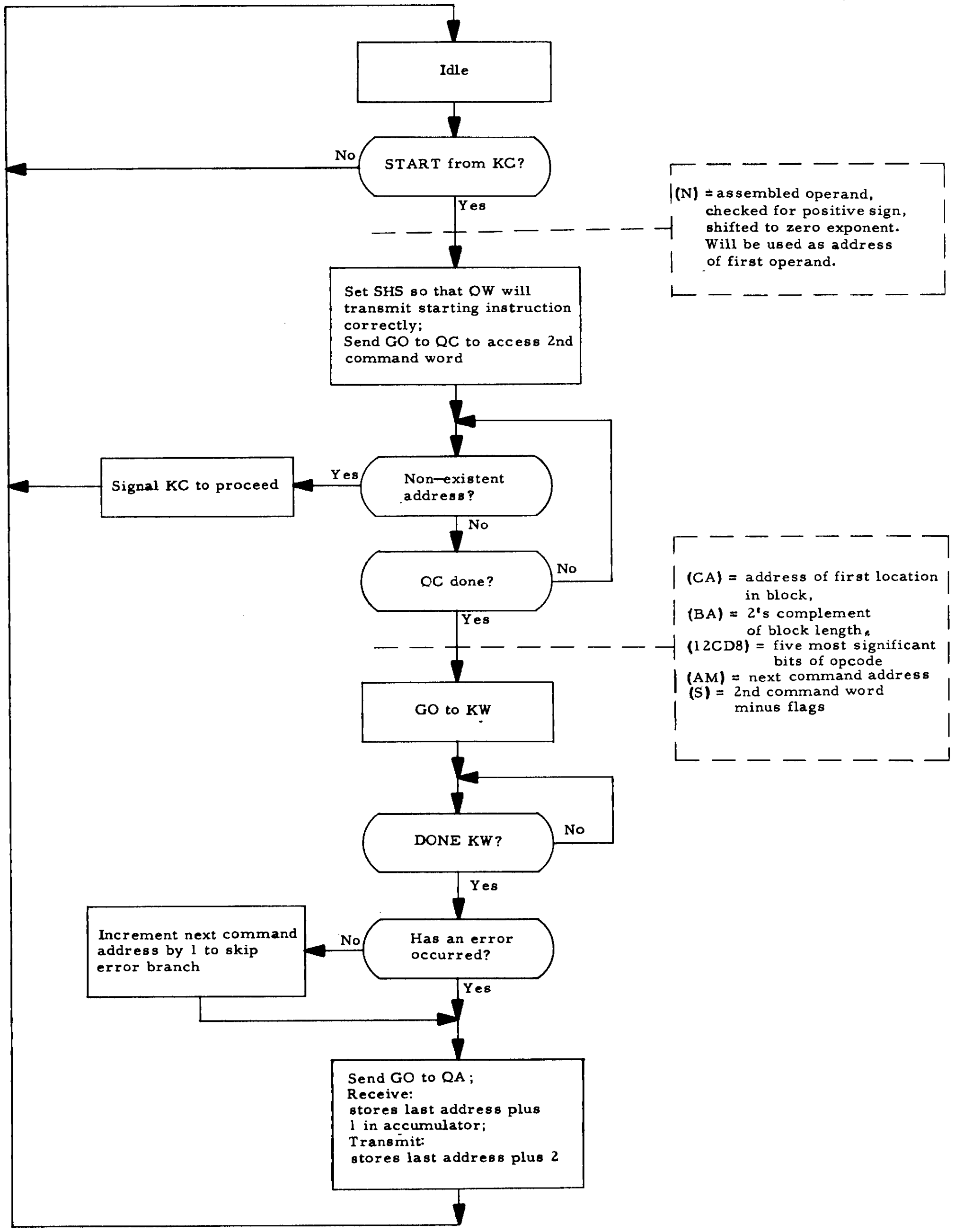
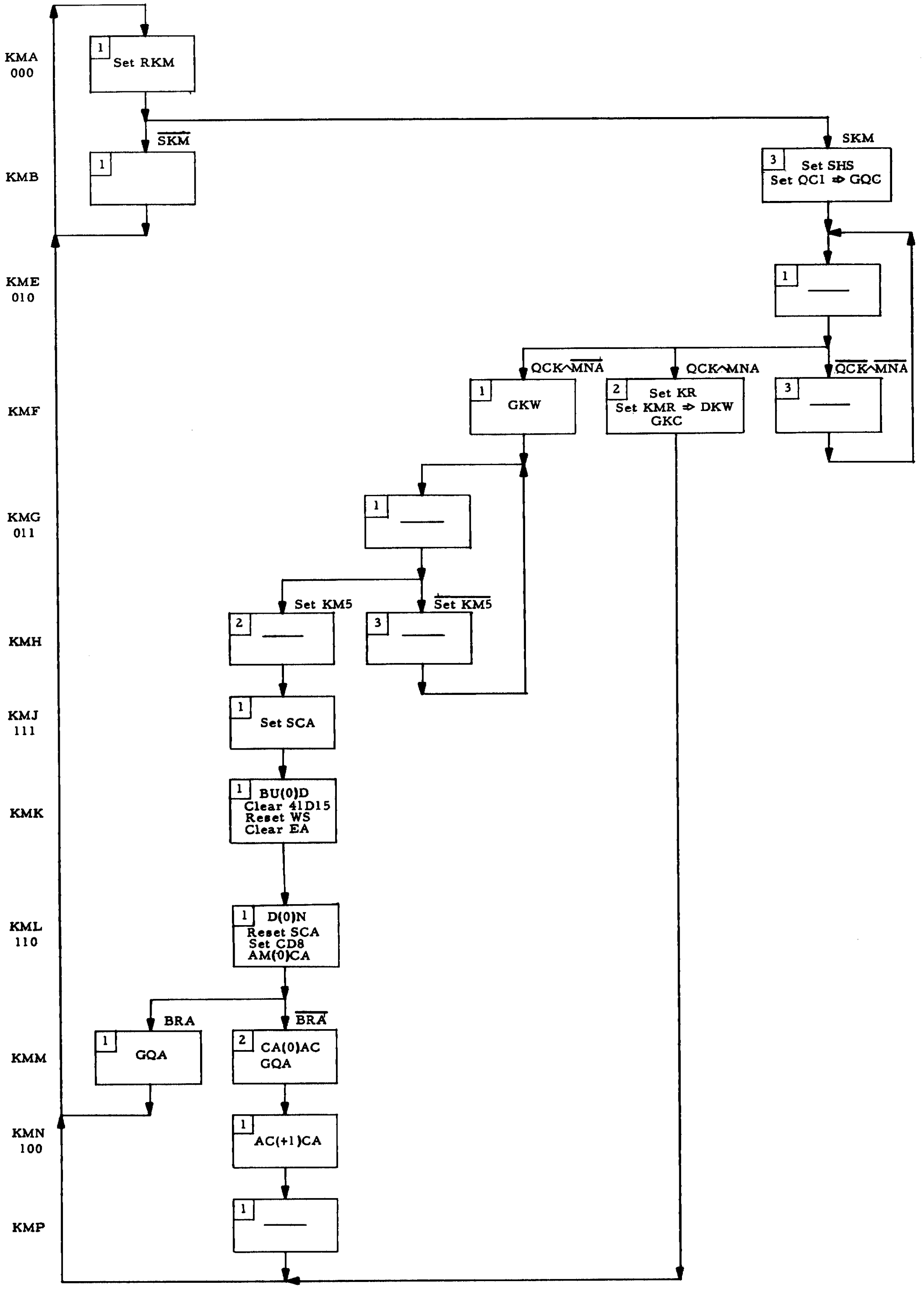

Idle
START from KC?
No
Yes
(N) = assembled operand, checked for positive sign, shifted to zero exponent. Will be used as address of first operand.
Set SHS so that QW will transmit starting instruction correctly;
Send GO to QC to access 2nd command word
Signal KC to proceed
Yes
Non—existent address?
No
QC done?
No
Yes
(CA) = address of first location in block,
(BA) = 2's complement of block length,
(12CD8) = five most significant bits of opcode
(AM) = next command address
(S) = 2nd command word minus flags
GO to KW
DONE KW?
No
Yes
Increment next command address by 1 to skip error branch
No
Has an error occurred?
Yes
Send GO to QA;
Receive:
stores last address plus 1 in accumulator;
Transmit:
stores last address plus 2

TABLE 12.5–1 Terms Used on the KM Flow Chart for Control of Block Input/Output

AM(0)CA	Path enabled to restore the next command address to CA register.
BRA	BRAnch flip–flop; BRA is set when ER is high; the input/output program will branch on execution of the next command.
BU(0)D	Path enabled to send the current information on the bus to register D.
Clear EA	Register EA holds the exponent of the information in register N; since QA will store an address from the N register into the Accumulator, and the address should have a zero exponent, it is necessary to clear EA to insure against attachment of an erroneous exponent to the address.
GKW	GO signal to KW sequencer.
GQA	GO to the QA sequencer.
GQC	GO to QC; there is actually no such signal; QC is started when QM sets the QC1 flip–flop.
KR	Opcode DONE signal to KC.
Reset WS	Working Sign must be positive to agree with address information to be stored by QA.
SCA	Select bus register CA for transfer of information; this will make possible the storing of the address of the last operand plus one in the Accumulator.
Set CD8	Command decoding bit set to enable interrupt processing; T0 looks like N0 without 7CD6 and an interrupt won't be processed following an N command.
Set KM5	KW signals DONE to KM by advancing KM logic (setting KM5 sends KM to KMJ).
SHS	Shift–Send mode; SHS high directs QW to shift a single character to transmitting position and to send it.
QCK	Decoding for state QCK of QC sequencer; when QC reaches this point in its processing, it is almost finished and KM considers this the signal to continue.

FIGURE 12.5–2 The KM Flow Chart for the Control of Block Input/Output

KM



SECTION 12.6 – THE QC SEQUENCER

QC has a single responsibility: that of handling the second command word in multiple access operations. It is started by KM to access this word and set up the information contained therein. In the case of block input/output operations, this word has the form:

```
 31    27      22                   14            0
| 0000 | opcode | starting instruction | block length |
```

In repeat arithmetic or logic operations, the word has the form

```
 31    27      20                   14            0
| 0000 | opcode |        000000        | block length |
```

These formats differ in that repeat operations do not call for inclusion of a starting instruction and the entire 7–bit opcode is specified in the repeat operation, whereas only 5 bits of the opcode are used in the block command. Neither of these differences are noticed by QC. The starting instruction in the block operation is handled elsewhere and only 5 bits of the opcode are preserved for decoding purposes in the repeat operations, these being sufficient to differentiate between the 32 repeat commands, so that no distinction need be made by QC.

When started, QC uses QM to access the second command word. When the word is available, the following actions are taken:

1) the next command address held in CA is sent to the AM register for the duration of the block operation and the address of the first operand (the assembled operand from the first command word) is sent to CA. With the operand address in CA, accesses are carried out by QM in the normal manner with the address in CA being incremented each time QM is started. The next command address is restored to CA by the KM sequencer when the block is complete.

2) The five most significant bits of the opcode are sent to CD for dècoding.

3) The block length is stored in register BA. Some explanation is required regarding the handling of the block length. The most obvious way to keep track of the current stage of the operation would be to decrement the block length each time an operand access occurred and test the remaining length against zero. A zero result would, of course, indicate the end of the block. However, the fact that the BA register is not used for operand addresses during block operations makes it available for storage of the block length and, in addition, makes available the increment paths associated with BA. (When QM is started by the signal GBA, the address in BA is incremented by means of the BA(0)AC, AC(+1)BA paths.) Thus, if the block length is stored in BA in complement form, these increment paths can be used to keep track of the progress of the operation and no additional logic is required. QC stores the block length in BA in 2's complement form, this because the addition of any number to its 2's complement results in a string of zeros of the same length with the addition of a leading 1 bit. For example, the 2's complement of the number 11 = 01.

$$\begin{array}{r} 11 \\ \underline{+01} \\ 100 \end{array}$$

Thus, the end of the block is detected by the change of state of the leading bit. Since the number of words available in memory will never exceed 32768_{10}, the block length can be specified in the address portion of the second command word.

The BA register is 16 bits in length (15BA0). QC transfers the 1's complement of the block length into 14BA0, leaving the

BA15 flip-flop reset so that it can be used to signal the end of the block. BA15 is set when ZBA is high, indicating that the AC register contains all 1's and the path AC(+1)BA is enabled. This will cause termination of the block operation. As an example of the handling of block length, consider what would occur in the handling of a block length of 17_8. This would appear in the address portion of the command word as

000 000 000 001 111

QC would complement the command word and send 14N0 to BA. This transfer does not affect the reset condition of BA15. The contents of BA would then be

0 111 111 111 110 000

Enabling of the increment path by QC leaves the 2's complement in BA

0 111 111 111 110 001

17_8 increments later, if no enabled flags or error conditions occur to end the operation earlier, the contents of BA will be

1 000 000 000 000 000

and the operation will be terminated.

FIGURE 12.6–1 Algorithm for Accessing the Second Command Word for Repeat and Block Input/Output Operations

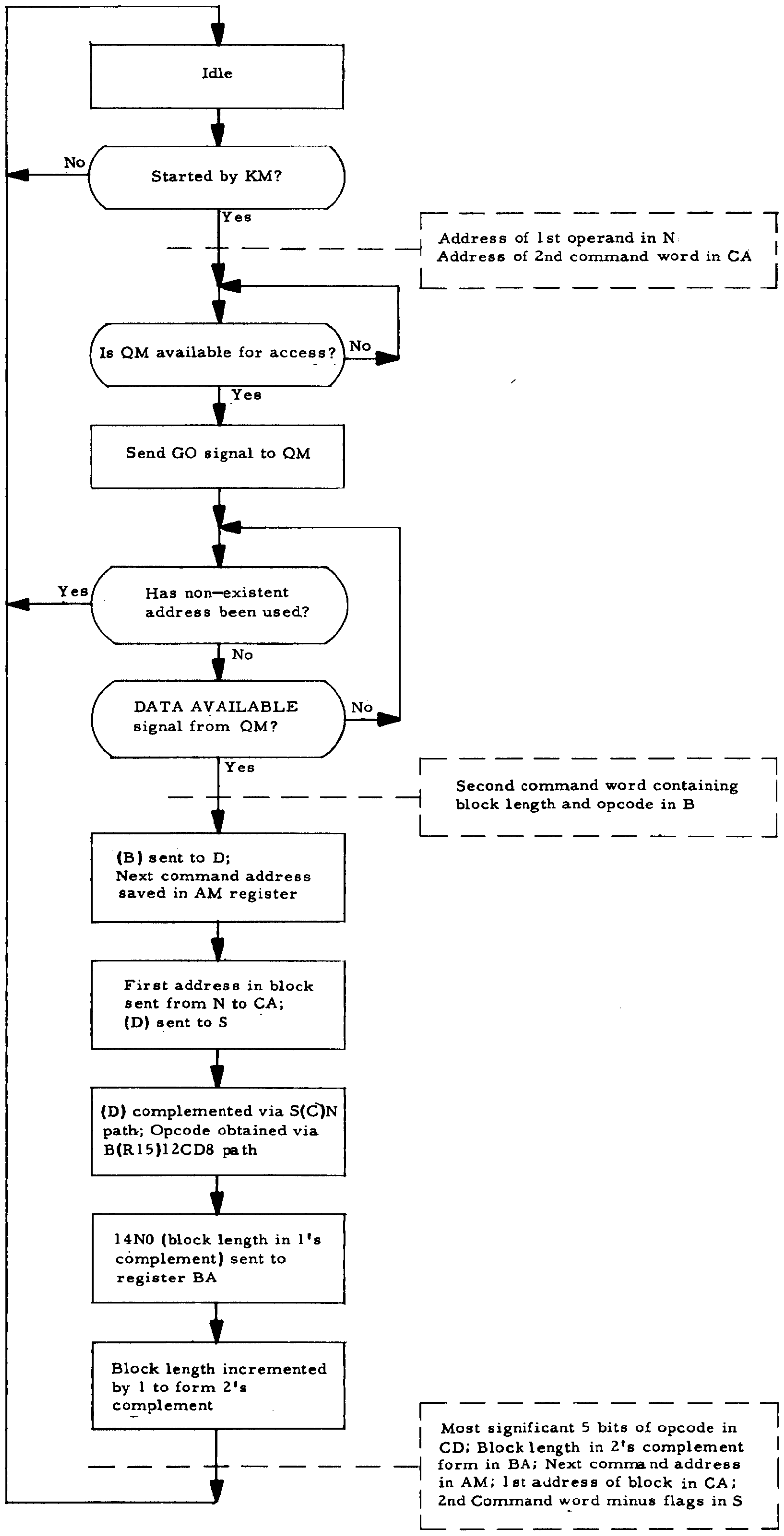
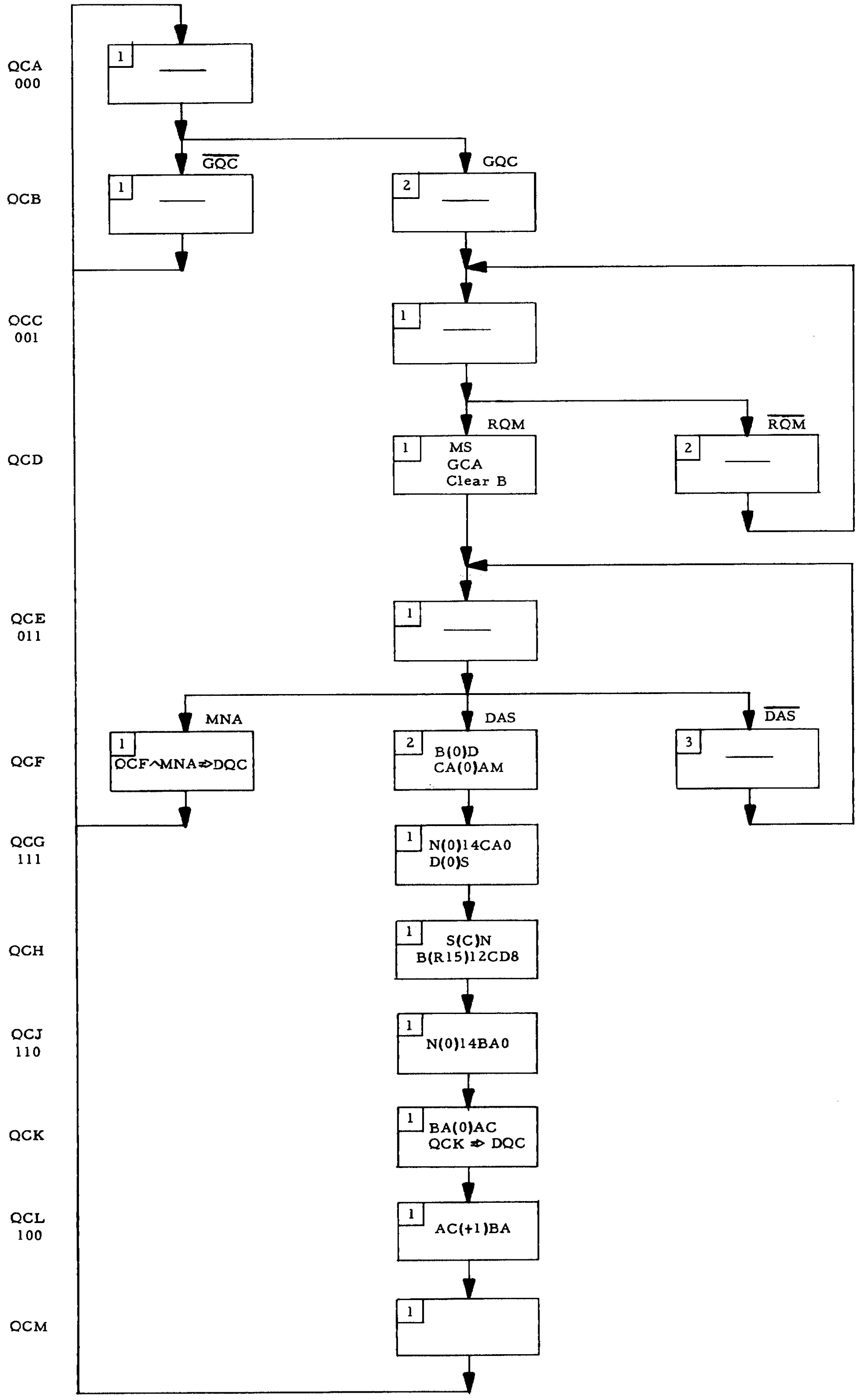

Idle
Started by KM?
No
Yes
Address of 1st operand in N
Address of 2nd command word in CA
Is QM available for access?
No
Yes
Send GO signal to QM
Has non–existent address been used?
Yes
No
DATA AVAILABLE signal from QM?
No
Yes
Second command word containing block length and opcode in B
(B) sent to D; Next command address saved in AM register
First address in block sent from N to CA; (D) sent to S
(D) complemented via S(C)N path; Opcode obtained via B(R15)12CD8 path
14N0 (block length in 1's complement) sent to register BA
Block length incremented by 1 to form 2's complement
Most significant 5 bits of opcode in CD; Block length in 2's complement form in BA; Next command address in AM; 1st address of block in CA; 2nd Command word minus flags in S

TABLE 12.6–1 Terms Used on the QC Flow Chart

DAS	Data Available Signal from QM; DAS is high when the accessed operand is in register B.
DQC	DONE QC sequencer; DQC is a non–existent signal; KM decodes either QCF ∧ MNA or QCK for DQC.
GCA	Go to QM sequencer; GCA ⇒ CA(0)AC, CA(0)MA.
GQC	Go to QC; non–existent signal, QC is started when KM sets the QC1 flip–flop.
MNA	Memory Non–existent Address.
RQM	READY QM sequencer.

FIGURE 12.6–2 The QC Flow Chart

QC



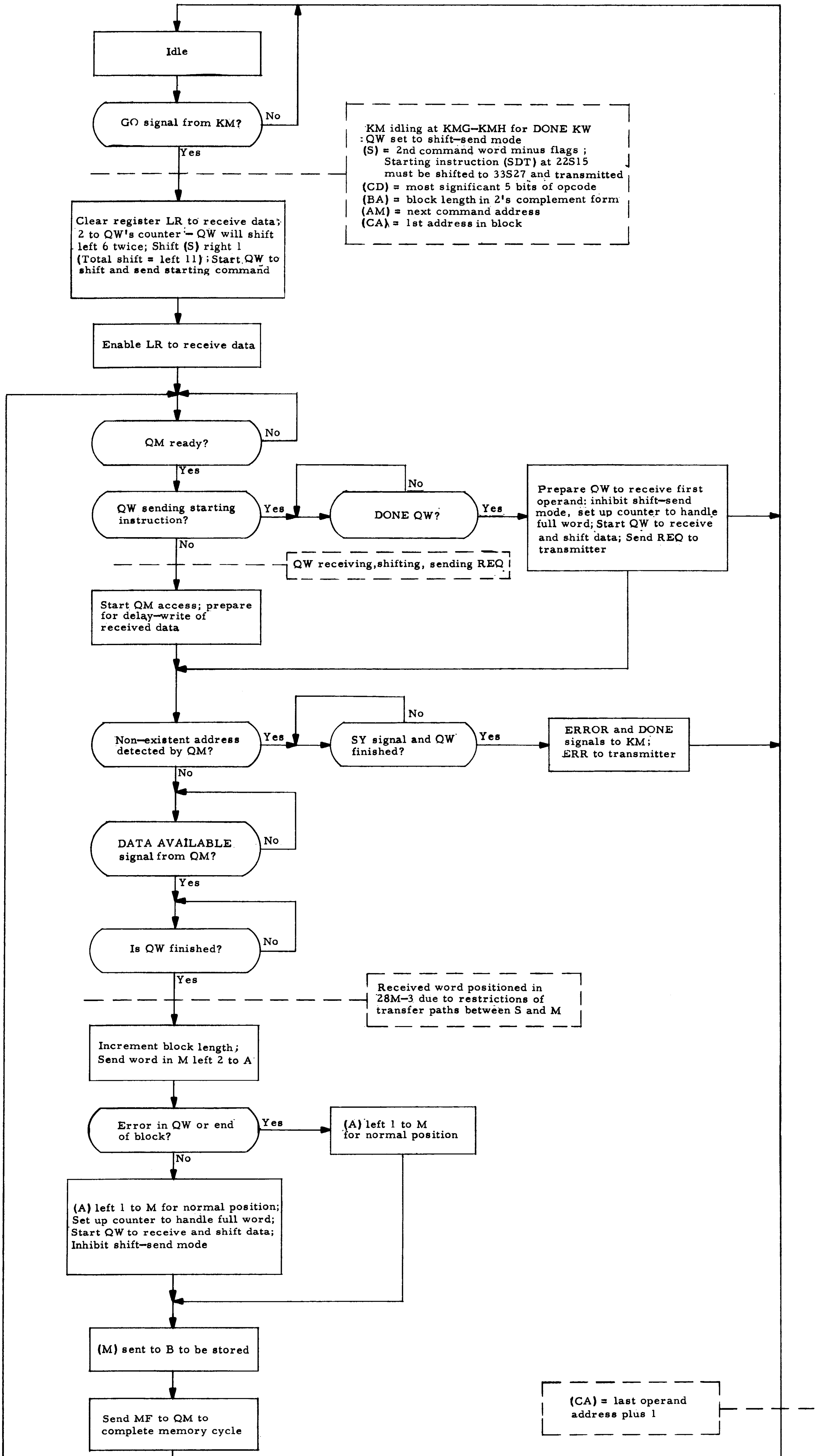
SECTION 12.7 – THE KW SEQUENCER

Perhaps the most involved aspect of the operation of KW is that of timing. KW functions simultaneously with KM, QM, and QW and is the controller for their combined activities. KM does not require attention; it idles until the block operation is terminated. QW and QM must both be started in order to process each word. The timing of these starts varies depending upon whether the operation is transmit or receive and whether or not this is the first operand. In general, QM is always one operand ahead of QW on transmit operations so that the next operand will be available for transmission. On receive, QW is slightly ahead of QM, but works on the same operand for part of the cycle. Receive operations use the memory in the delay–write mode. Thus, QM reads out the location, signals KW, and waits to be told that it can go ahead with the write half of the cycle. KW also waits for QW to finish the receipt of the word, then transfers the word to the B register and signals QM to finish the cycle. In the meantime, QW can begin receiving the next word.

The fact that QM is ahead of QW in transmit, and follows it in receive, accounts for what gets stored in the Accumulator at the end of the operation. QA is started by KM to store the contents of the CA register as the last task in the block operation. The contents of CA will be the address of the last operand plus two in transmit, while in receive it will be the last address plus one. (Address incrementing occurs when QM is started up.) To avoid confusion, the block transmit algorithm calls for an extra increment of the address when the operation terminates due to the end of block signal because, in this case, no access was made the last time through the algorithm. If this additional increment were not included, the address stored would be the last plus one when termination was due to the end of the block, and the address plus two when termination was due to the occurrence of an error.

It is interesting to note that, at termination of the operation, the END or ERR code will be sent to the other device on BRD, and BTD operations, but never on BTC, since END and ERR are used to terminate block data transfers, but are meaningless following a string of line commands.

FIGURE 12.7–1 The KW Algorithm for Block Receive


Idle
GO signal from KM?
No
Yes
KM idling at KMG–KMH for DONE KW
: QW set to shift–send mode
(S) = 2nd command word minus flags ;
Starting instruction (SDT) at 22S15
must be shifted to 33S27 and transmitted
(CD) = most significant 5 bits of opcode
(BA) = block length in 2's complement form
(AM) = next command address
(CA) = 1st address in block
Clear register LR to receive data; 2 to QW's counter – QW will shift left 6 twice; Shift (S) right 1 (Total shift = left 11); Start QW to shift and send starting command
Enable LR to receive data
QM ready?
No
Yes
QW sending starting instruction?
Yes
No
DONE QW?
No
Yes
Prepare QW to receive first operand: inhibit shift–send mode, set up counter to handle full word; Start QW to receive and shift data; Send REQ to transmitter
QW receiving, shifting, sending REQ
Start QM access; prepare for delay–write of received data
Non–existent address detected by QM?
Yes
No
SY signal and QW finished?
No
Yes
ERROR and DONE signals to KM; ERR to transmitter
DATA AVAILABLE signal from QM?
No
Yes
Is QW finished?
No
Yes
Received word positioned in 28M–3 due to restrictions of transfer paths between S and M
Increment block length; Send word in M left 2 to A
Error in QW or end of block?
Yes
No
(A) left 1 to M for normal position
(A) left 1 to M for normal position; Set up counter to handle full word; Start QW to receive and shift data; Inhibit shift–send mode
(M) sent to B to be stored
Send MF to QM to complete memory cycle
(CA) = last operand address plus 1

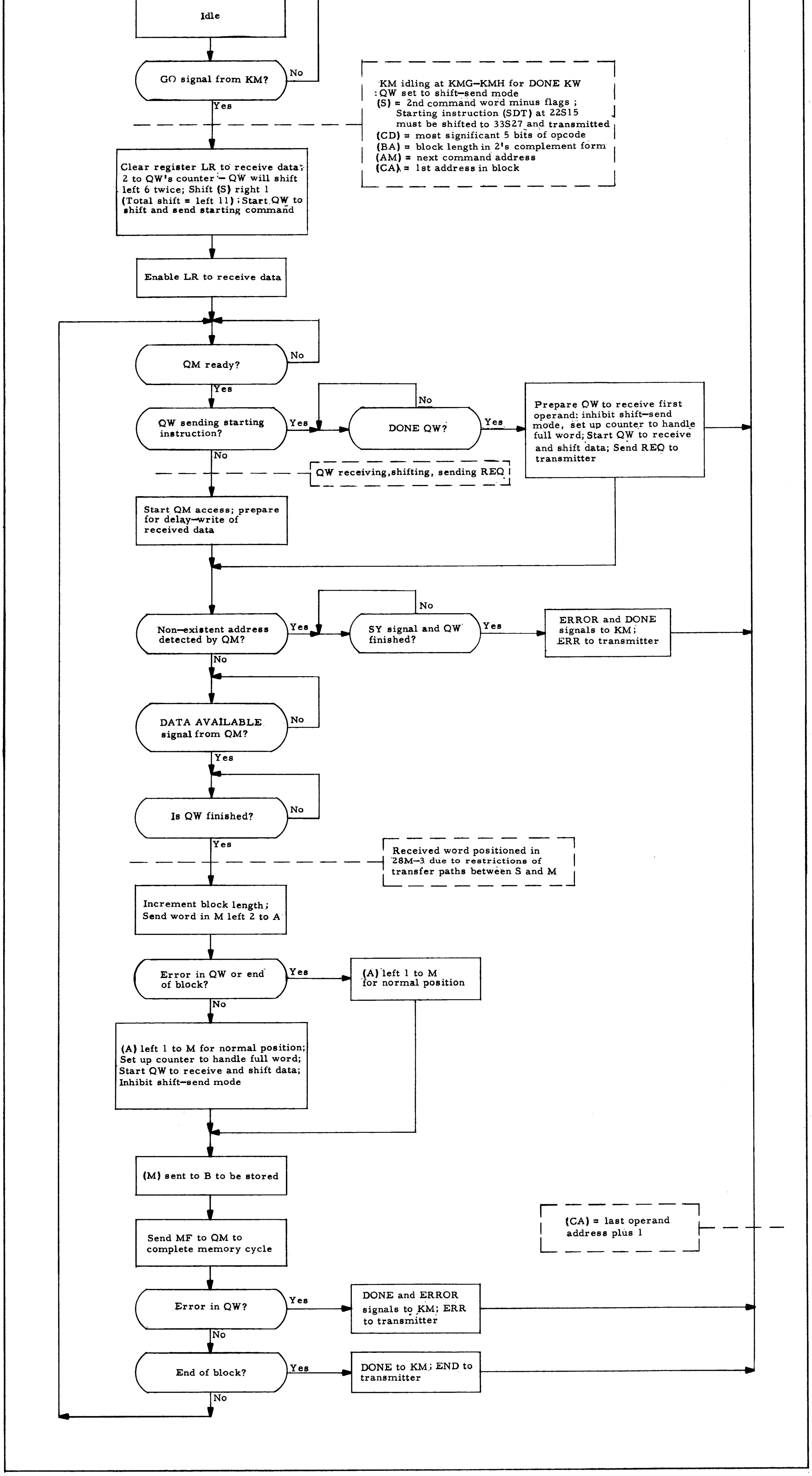

Idle
GO signal from KM?
No
Yes
KM idling at KMG–KMH for DONE KW
:QW set to shift–send mode
(S) = 2nd command word minus flags ;
Starting instruction (SDT) at 22S15
must be shifted to 33S27 and transmitted
(CD) = most significant 5 bits of opcode
(BA) = block length in 2's complement form
(AM) = next command address
(CA) = 1st address in block
Clear register LR to receive data;
2 to QW's counter – QW will shift
left 6 twice; Shift (S) right 1
(Total shift = left 11) ; Start QW to
shift and send starting command
Enable LR to receive data
QM ready?
No
Yes
QW sending starting instruction?
Yes
DONE QW?
No
Yes
Prepare QW to receive first operand: inhibit shift–send mode, set up counter to handle full word; Start QW to receive and shift data; Send REQ to transmitter
No
QW receiving,shifting, sending REQ
Start QM access; prepare for delay–write of received data
Non–existent address detected by QM?
Yes
SY signal and QW finished?
No
Yes
ERROR and DONE signals to KM; ERR to transmitter
No
DATA AVAILABLE signal from QM?
No
Yes
Is QW finished?
No
Yes
Received word positioned in 28M–3 due to restrictions of transfer paths between S and M
Increment block length; Send word in M left 2 to A
Error in QW or end of block?
Yes
(A) left 1 to M for normal position
No
(A) left 1 to M for normal position; Set up counter to handle full word; Start QW to receive and shift data; Inhibit shift–send mode
(M) sent to B to be stored
Send MF to QM to complete memory cycle
(CA) = last operand address plus 1
Error in QW?
Yes
DONE and ERROR signals to KM; ERR to transmitter
No
End of block?
Yes
DONE to KM; END to transmitter
No

TABLE 12.7–1 Terms Used on the KW Flow Chart for Block Receive

BLA	Block Line Amplifier flip–flop; when set, blocks inputs to the Line Receiver register.
BRA	BRAnch flip–flop; when set, indicates that ER is high due to either the occurrence of an error or the occurrence of a condition that calls for a branch in the program; computation continues with the next command.
Clear CQ	Set up CP–CQ counter to handle appropriate shift in SHS mode; for starting instructions, KW sets the counter to 011 to effect two shifts of 6 bits each.
DKW	DONE KW sequencer; signal sent to the KM sequencer when KW is finished.
DW	Delay–Write flip–flop; when set, causes the memory cycle to wait at the end of the read part of the cycle until a Memory Finish signal is supplied.
END(0)LD ERR(0)LD	The end of the block operation is signalled to the other device by means of transmission of one of these commands if the operation is not BTC6 or BTC8.
GCA	GO signal to the QM sequencer; GCA ⇒ CA(0)MA, CA(0)AC.
Fix CQ	Set up CP–CQ counter to handle a full word (000 if receiving 6–bit mode, 001 if 8–bit).
GQW	GO to QW sequencer.
LR	Line Receiver register.
MF	Memory Finish flip–flop; set by KW when the word to be stored in memory has been positioned in the B register.
RET	RETurn flip–flop; set after handling of first operand in block operation to gate KW correctly.
RL	Reset Line receiver register flip–flop; Set RL ⇒ reset LR, reset BLA.
RQM	READY QM sequencer.
RQW	READY QW sequencer.
SHS	Shift–Send flip–flop; used in block operations only for transmission of starting instruction; reset throughout the remainder of the block operation.
SY	SYnchronization flip–flops.
TRF	TRansmit request For character flip–flop; when TRF is set, LD14 transmits an REQ signal on the separate REQ line in the communication cable.

FIGURE 12.7–2 The KW Flow Chart for Block Receive

KW

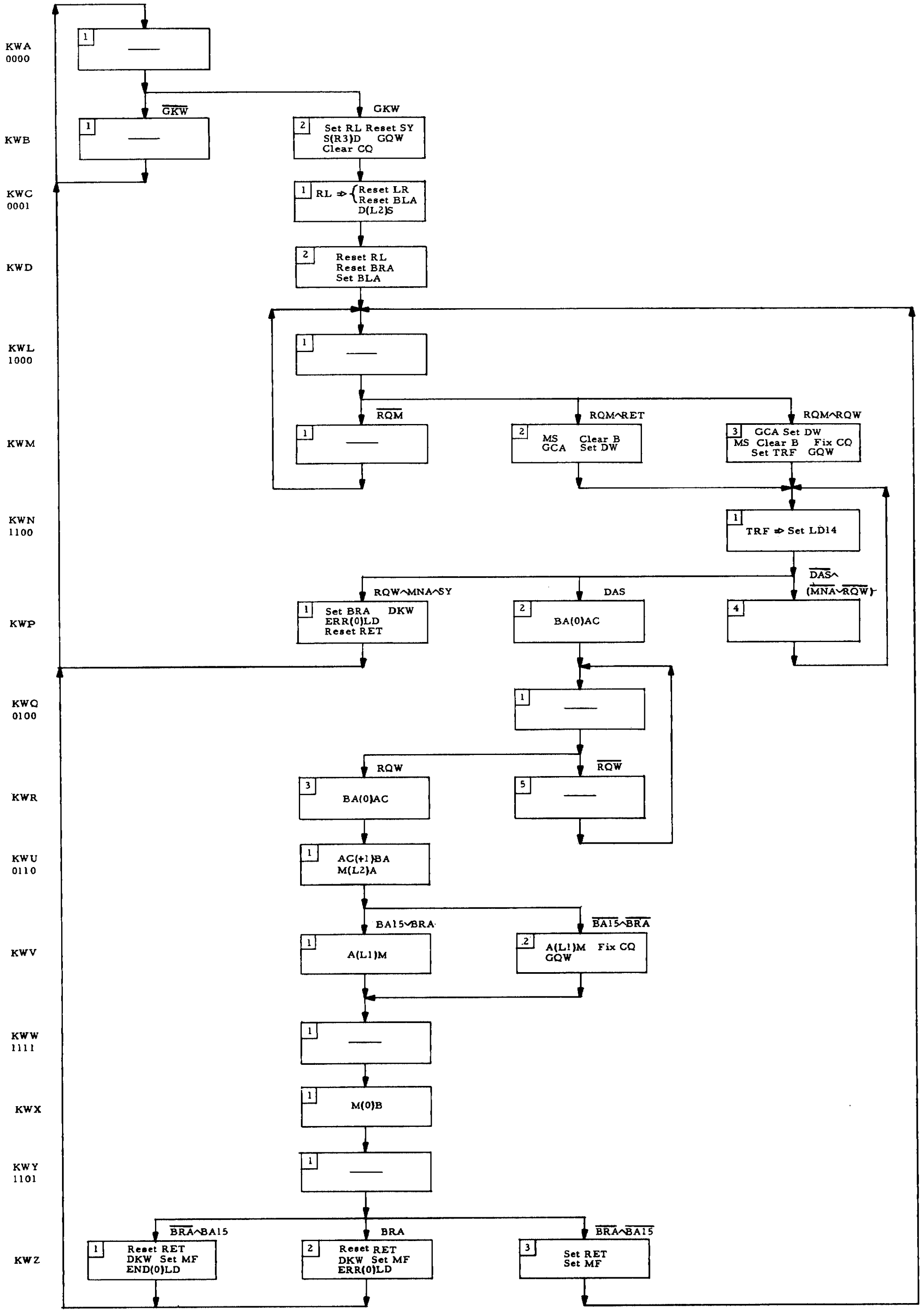
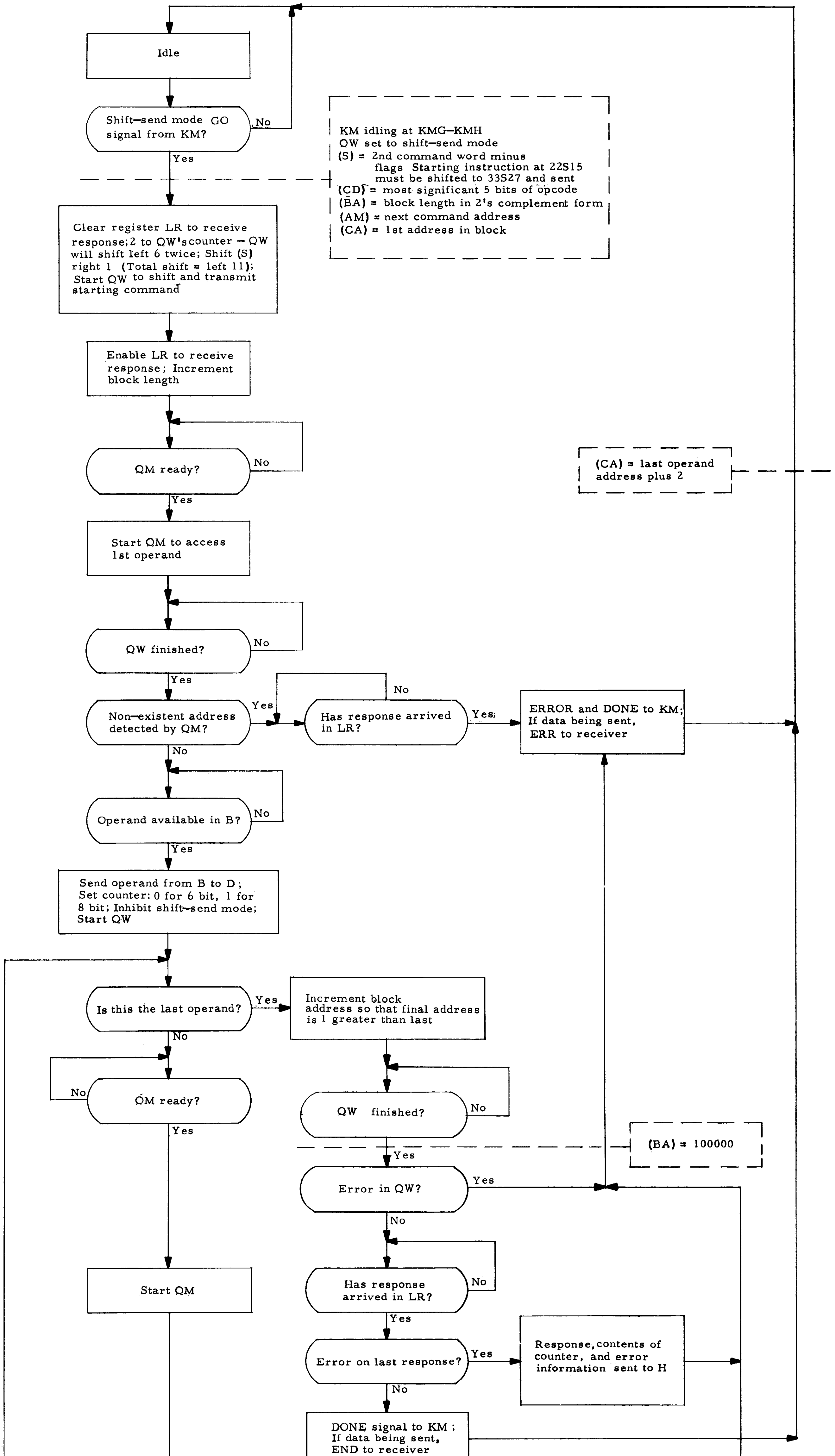


FIGURE 12.7–3 The KW Algorithm for Block Transmit


Idle
Shift–send mode GO signal from KM?
No
Yes
KM idling at KMG–KMH
QW set to shift–send mode
(S) = 2nd command word minus flags Starting instruction at 22S15 must be shifted to 33S27 and sent
(CD) = most significant 5 bits of opcode
(BA) = block length in 2's complement form
(AM) = next command address
(CA) = 1st address in block
Clear register LR to receive response;2 to QW'scounter – QW will shift left 6 twice; Shift (S) right 1 (Total shift = left 11); Start QW to shift and transmit starting command
Enable LR to receive response; Increment block length
QM ready?
No
Yes
(CA) = last operand address plus 2
Start QM to access 1st operand
QW finished?
No
Yes
Non–existent address detected by QM?
Yes
No
Has response arrived in LR?
No
Yes
ERROR and DONE to KM; If data being sent, ERR to receiver
Operand available in B?
No
Yes
Send operand from B to D; Set counter: 0 for 6 bit, 1 for 8 bit; Inhibit shift–send mode; Start QW
Is this the last operand?
Yes
No
Increment block address so that final address is 1 greater than last
QM ready?
No
Yes
QW finished?
No
Yes
(BA) = 100000
Error in QW?
Yes
No
Start QM
Has response arrived in LR?
No
Yes
Error on last response?
Yes
No
Response, contents of counter, and error information sent to H
DONE signal to KM ; If data being sent, END to receiver

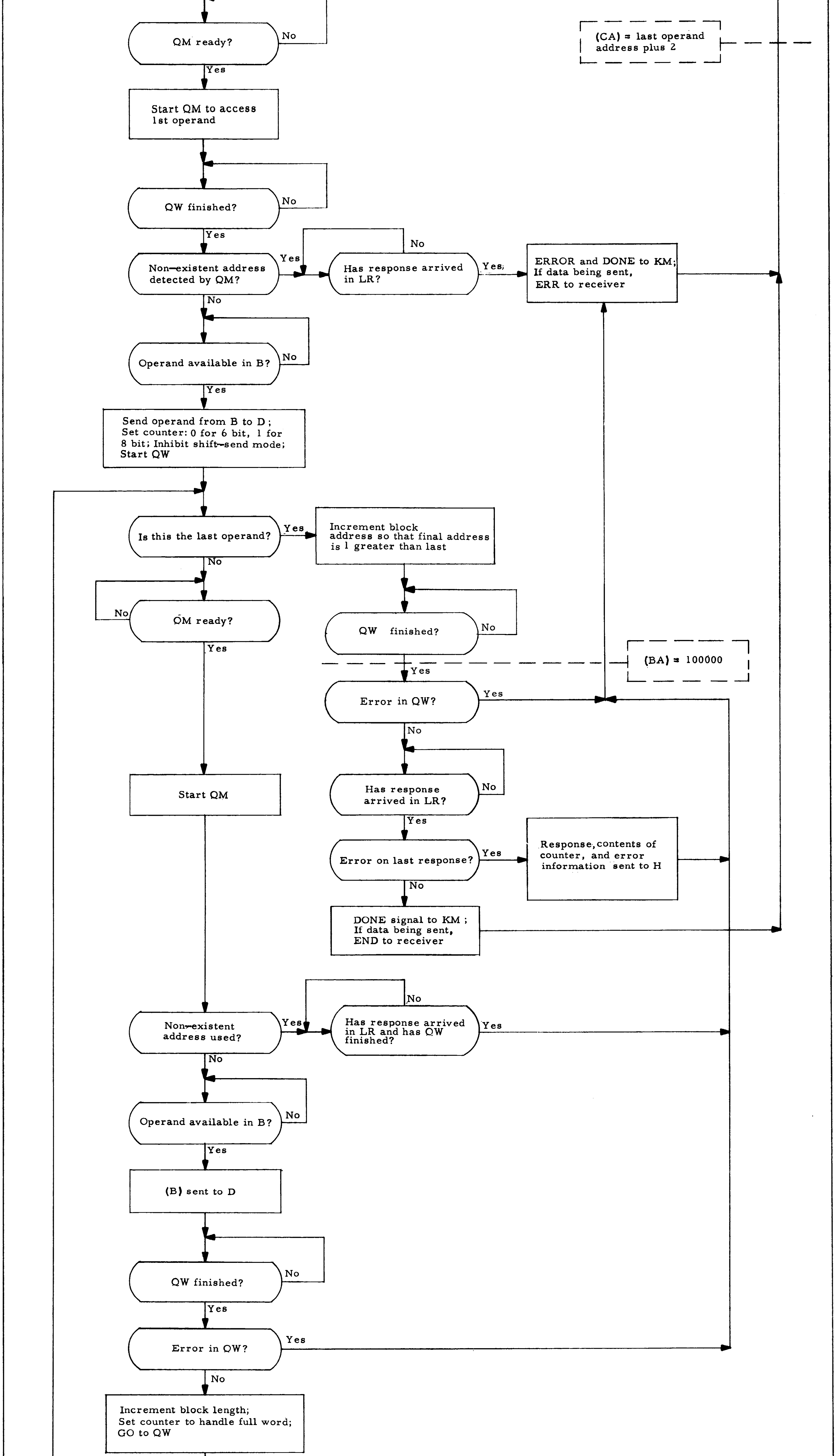
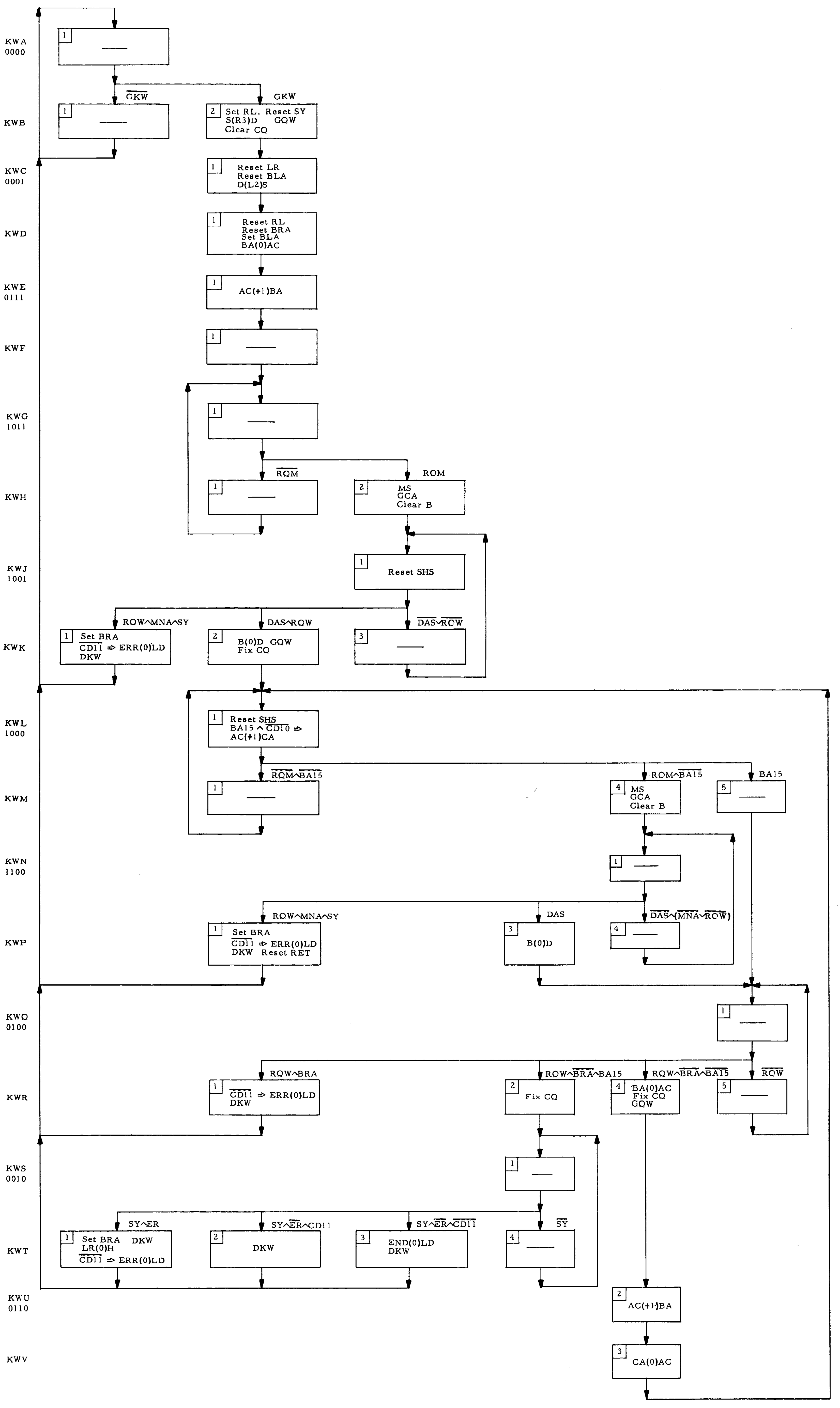

QM ready?
No
Yes
(CA) = last operand address plus 2
Start QM to access 1st operand
QW finished?
No
Yes
Non–existent address detected by QM?
Yes
Has response arrived in LR?
No
Yes;
ERROR and DONE to KM; If data being sent, ERR to receiver
No
Operand available in B?
No
Yes
Send operand from B to D; Set counter: 0 for 6 bit, 1 for 8 bit; Inhibit shift–send mode; Start QW
Is this the last operand?
Yes
Increment block address so that final address is 1 greater than last
No
No
QM ready?
QW finished?
No
Yes
(BA) = 100000
Yes
Error in QW?
Yes
No
Has response arrived in LR?
No
Yes
Start QM
Error on last response?
Yes
Response, contents of counter, and error information sent to H
No
DONE signal to KM ; If data being sent, END to receiver
Non–existent address used?
Yes
Has response arrived in LR and has QW finished?
No
Yes
No
Operand available in B?
No
Yes
(B) sent to D
QW finished?
No
Yes
Error in QW?
Yes
No
Increment block length; Set counter to handle full word; GO to QW

TABLE 12.7–2 Terms Used on the KW Flow Chart for Block Transmit

BLA	Block Line Amplifier flip–flop; when set, blocks inputs to the Line Receiver register.
BRA	BRAnch flip–flop; when set, indicates that ER is high due to either the occurrence of an error or the occurrence of a condition that calls for a branch in the program; computation continues with the next command.
CD11	Command decoding bit used to distinguish between types of transmission; CD11 high indicates command transmission, low indicates data.
Clear CQ	Set up CP–CQ counter to handle appropriate shift in SHS mode; for starting instruction, KW sets the counter to 011 to effect two shifts of 6 bits each.
DKW	DONE KW sequencer; signal sent to the KM sequencer when KW is finished.
Fix CQ	Set up CP–CQ counter to handle a full word (000 if receiving 6–bit mode, 001 if 8–bit).
GCA	GO signal to the QM sequencer; GCA ⇒ CA(0)MA, CA(0)AC.
END(0)LD ERR(0)LD	The end of the block operation is signalled to the other device by means of transmission of one of these commands if the operation is not BTC6 or BTC8.
GQW	GO to QW sequencer.
LR	Line Receiver register; flip–flops in LR must be reset before response can be received.
LR(0)H	Path enabled when ER is high to send last response received to H register.
RET	RETurn flip–flop; set after handling of first operand in block operation to gate KW correctly.
RL	Reset Line receiver register flip–flop; Set RL ⇒ reset LR, reset BLA.
RQM	READY QM sequencer.
RQW	READY QW sequencer.
SHS	Shift–Send flip–flop; used in block operations only for transmission of starting instruction; reset throughout the remainder of the block operation.
SY	SYnchronization flip–flops.

FIGURE 12.7–4 The KW Flow Chart for Block Transmit

KW



12.7

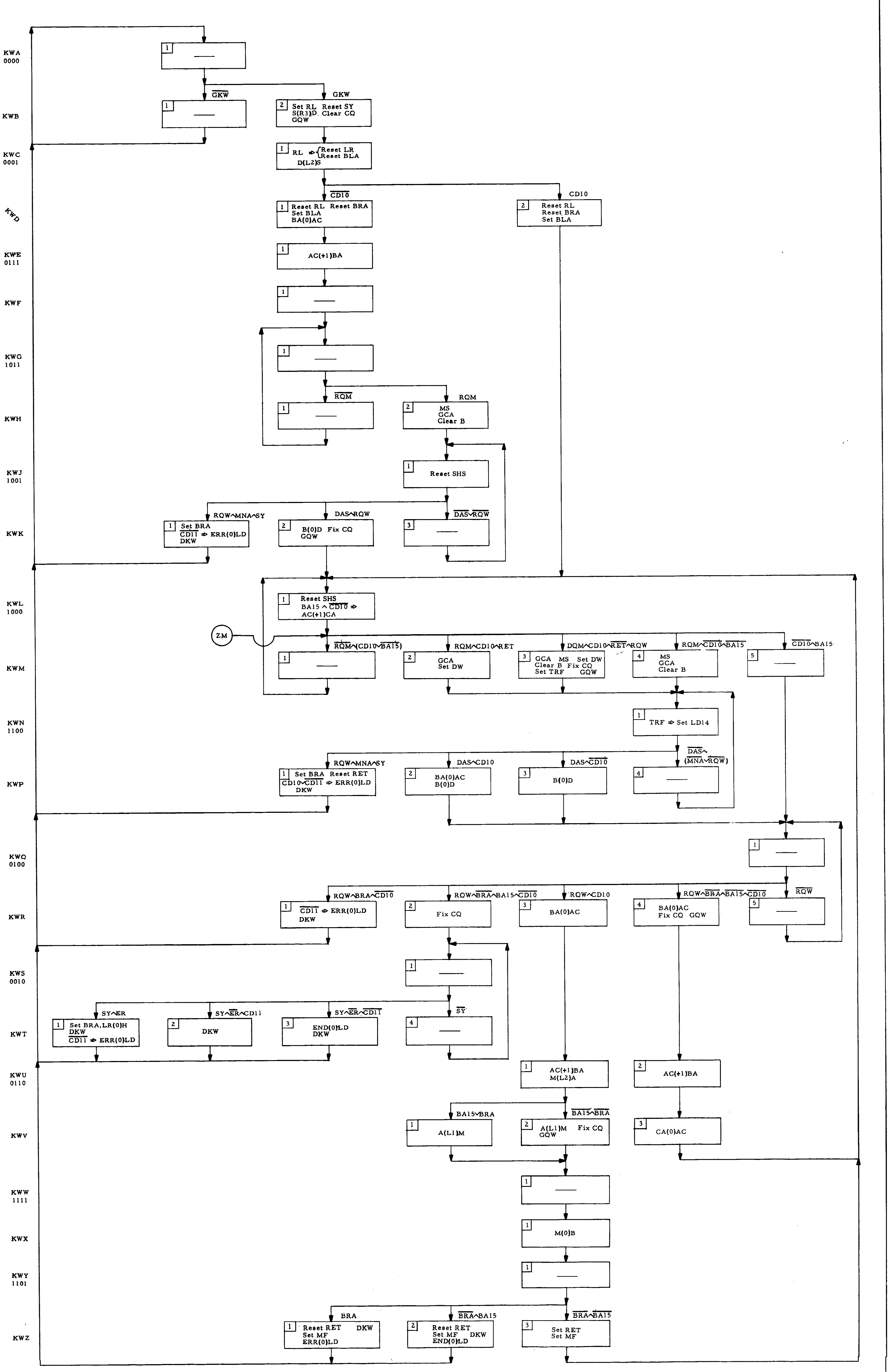
TABLE 12.7–3 Terms Used on the KW Flow Chart

All those shown separately for transmit and receive with the additions:

Command decoding bit designations:

CD10	receive operation,
$\overline{\mathrm{CD10}}$	transmit,
$\overline{\mathrm{CD10}} \wedge \mathrm{CD11}$	transmit commands,
ZM	Zero Machine; this entry to KW is used in the bootstrap or initializing operation to allow input of the control program which is stored in location 64_{10}.

FIGURE 12.7–5 The KW Flow Chart


KWA 0000
KWB
KWC 0001
KWD
KWE 0111
KWF
KWG 1011
KWH
KWJ 1001
KWK
KWL 1000
KWM
KWN 1100
KWP
KWQ 0100
KWR
KWS 0010
KWT
KWU 0110
KWV
KWW 1111
KWX
KWY 1101
KWZ
$\overline{GKW}$
GKW
2 Set RL Reset SY S(R3)D Clear CQ GQW
1 RL ⇒ {Reset LR Reset BLA D(L2)S
$\overline{CD10}$
CD10
1 Reset RL Reset BRA Set BLA BA(0)AC
2 Reset RL Reset BRA Set BLA
1 AC(+1)BA
$\overline{RQM}$
RQM
2 MS GCA Clear B
1 Reset SHS
RQW∧MNA∧SY
DAS∧RQW
$\overline{DAS}$∧$\overline{RQW}$
1 Set BRA $\overline{CD11}$ ⇒ ERR(0)LD DKW
2 B(0)D Fix CQ GQW
1 Reset SHS BA15 ∧ $\overline{CD10}$ ⇒ AC(+1)CA
ZM
$\overline{RQM}$∧(CD10∨$\overline{BA15}$)
RQM∧CD10∧RET
DQM∧CD10∧$\overline{RET}$∧RQW
RQM∧$\overline{CD10}$∧$\overline{BA15}$
$\overline{CD10}$∧BA15
2 GCA Set DW
3 GCA MS Set DW Clear B Fix CQ Set TRF GQW
4 MS GCA Clear B
1 TRF ⇒ Set LD14
RQW∧MNA∧SY
DAS∧CD10
DAS∧$\overline{CD10}$
$\overline{DAS}$∧ ($\overline{MNA∧RQW}$)
1 Set BRA Reset RET CD10∧$\overline{CD11}$ ⇒ ERR(0)LD DKW
2 BA(0)AC B(0)D
3 B(0)D
RQW∧BRA∧$\overline{CD10}$
RQW∧$\overline{BRA}$∧BA15∧$\overline{CD10}$
RQW∧CD10
RQW∧$\overline{BRA}$∧$\overline{BA15}$∧$\overline{CD10}$
$\overline{RQW}$
1 $\overline{CD11}$ ⇒ ERR(0)LD DKW
2 Fix CQ
3 BA(0)AC
4 BA(0)AC Fix CQ GQW
SY∧ER
SY∧$\overline{ER}$∧CD11
SY∧$\overline{ER}$∧$\overline{CD11}$
$\overline{SY}$
1 Set BRA,LR(0)H DKW $\overline{CD11}$ ⇒ ERR(0)LD
2 DKW
3 END(0)LD DKW
1 AC(+1)BA M(L2)A
2 AC(+1)BA
BA15∨BRA
$\overline{BA15}$∧$\overline{BRA}$
1 A(L1)M
2 A(L1)M Fix CQ GQW
3 CA(0)AC
1 M(0)B
BRA
$\overline{BRA}$∧BA15
$\overline{BRA}$∧$\overline{BA15}$
1 Reset RET DKW Set MF ERR(0)LD
2 Reset RET Set MF DKW END(0)LD
3 Set RET Set MF

SECTION 12.8 – THE QW SEQUENCER

The functions carried out by QW have been described in some detail in the introduction to this chapter as well as in Section 3 which handles the Shift–Send mode of QW. (This mode of operation allows for handling of the TLC and TDC commands and for the transmission of the starting instruction in block operations.) QW is started for the actual block transmission or receive operation following the sending of the starting instruction and the turning off of the Shift–Send flip–flop. It is important to remember that a separate startup of QW is required for the handling of each word in the operation. The character counter is set by KW at each entry (000 for 6–bit mode, 001 for 8–bit mode) and QW will send or receive characters until CP3 indicates that a full word has been handled. Control is then returned to KW which operates simultaneously, using QM to store or access the operands, setting up the character counter, and returning control to KM when the operation is terminated.

On block receive, the transmitting device expects to receive an REQ signal from the Central Processor for each character that it sends out. This signal is sent on a separate line in the communication cable and is triggered by the setting of LD14. The Transmit Request for character Flip–Flop (TRF) controls LD14. KW is responsible for setting TRF the first time QW is started. For all subsequent characters in the operation, QW sets TRF. Thus, at QWF, TRF is set, even on the last character in the particular word, unless ZBA is high. (ZBA is the signal used to indicate that the next block length increment will send BA15 high, i.e., this is the last word in the block.) Occurrence of an error before a word is complete causes QW to fill the remaining character positions with zeros before exiting (using blocks QWE2 and QWF6).

For both transmit and receive operations, receipt of a correct, but unexpected response, or of a wrong response, causes ER to go high and thus sets the BRAnch flip–flop. Thus, the program will continue with execution of the next command which calls for a transfer. Note that each transmission of a character is accompanied by the clearing of the LR register and enabling of the response (resetting BLA).

FIGURE 12.8–1 The QW Algorithm for Block Receive

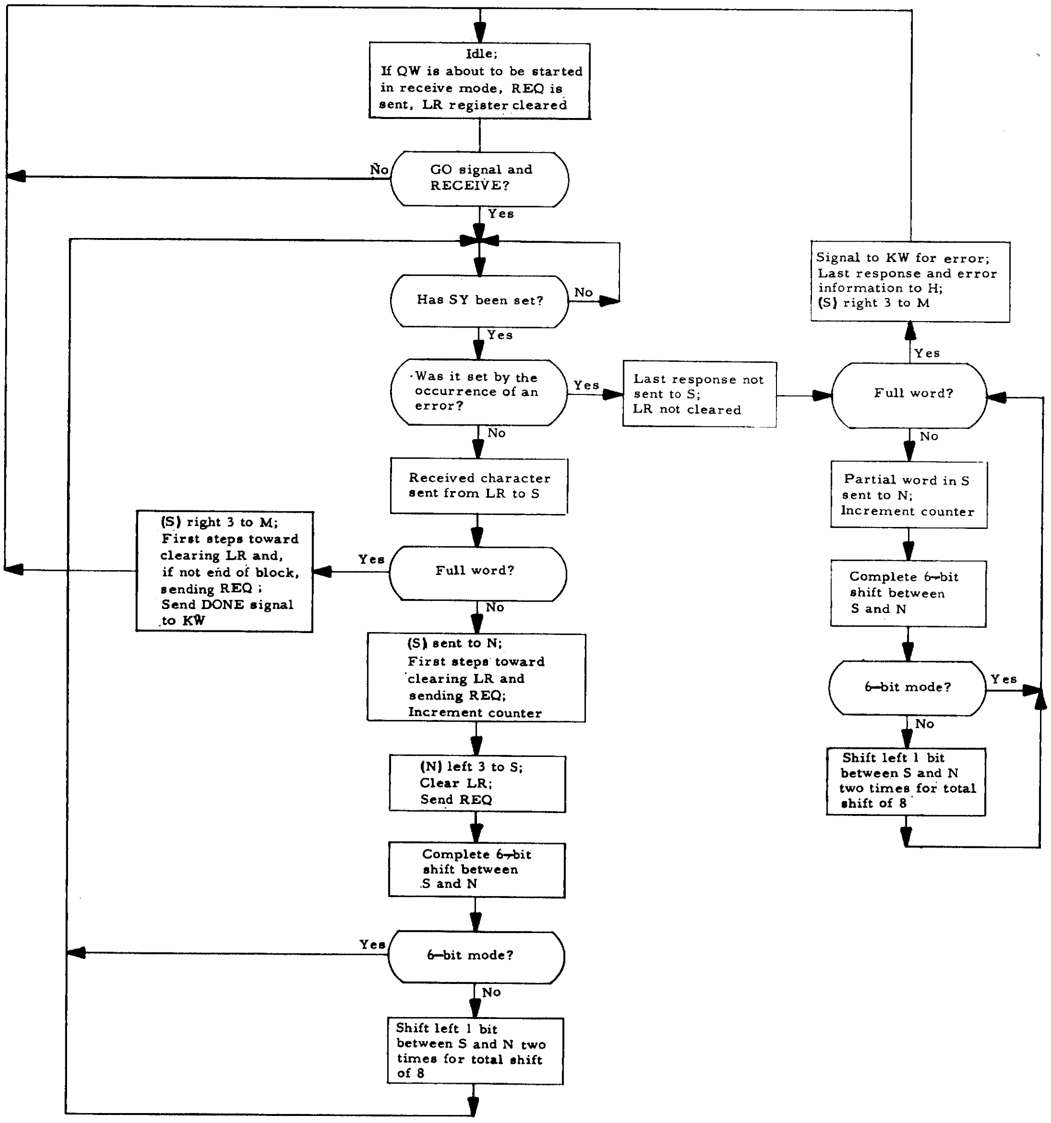

Idle;
If QW is about to be started in receive mode, REQ is sent, LR register cleared
GO signal and RECEIVE?
No
Yes
Has SY been set?
No
Yes
Was it set by the occurrence of an error?
Yes
No
Last response not sent to S; LR not cleared
Full word?
Yes
No
Signal to KW for error; Last response and error information to H; (S) right 3 to M
Partial word in S sent to N; Increment counter
Complete 6-bit shift between S and N
6-bit mode?
Yes
No
Shift left 1 bit between S and N two times for total shift of 8
Received character sent from LR to S
Full word?
Yes
No
(S) right 3 to M; First steps toward clearing LR and, if not end of block, sending REQ ; Send DONE signal to KW
(S) sent to N; First steps toward clearing LR and sending REQ; Increment counter
(N) left 3 to S; Clear LR; Send REQ
Complete 6-bit shift between S and N
6-bit mode?
Yes
No
Shift left 1 bit between S and N two times for total shift of 8

TABLE 12.8–1 Terms Used on the QW Flow Chart for Block Receive

(Most of these have been discussed elsewhere; short definitions are included at this point for ease of reference.)

BLA	Block Line receiver Amplifiers flip–flop.
BRA	BRAnch flip–flop.
CD10	Command decoding bit used to distinguish between transmit and receive operations; CD10 high indicates receive.
CD12	Command decoding bit used to distinguish between the 8 and 6–bit mode; CD12 high indicates 8–bit, low indicates 6–bit.
CP(+1)CQ	Increment path for CP–CQ character counter.
CP3	Signal sent by CP–CQ counter when a full word has been handled.
DQW	DONE QW sequencer.
ER	ERror signal.
LR(0)H	Path enabled when ER is high to send last response to the H register; (see Table 12.1–3 for explanation of transfers caused by enabling of LR(0)H).
LR(0)S	Path enabled to send the last response to the least significant bits of the S register.
LD14	When LD14 is high, a signal is sent on separate line in communication cable which is interpreted as a REQuest for character.
RL	Reset Line receiver register.
RQW	READY QW sequencer.
SY	SYnchronization flip–flop.
TRF	Transmit Request For character flip–flop.
ZBA	BA register Zero signal; ZBA is high when (AC) indicate that the next enabling of the AC(+1)BA increment path will cause BA15 to go high.

FIGURE 12.8–2 The QW Flow Chart for Block Receive

QW

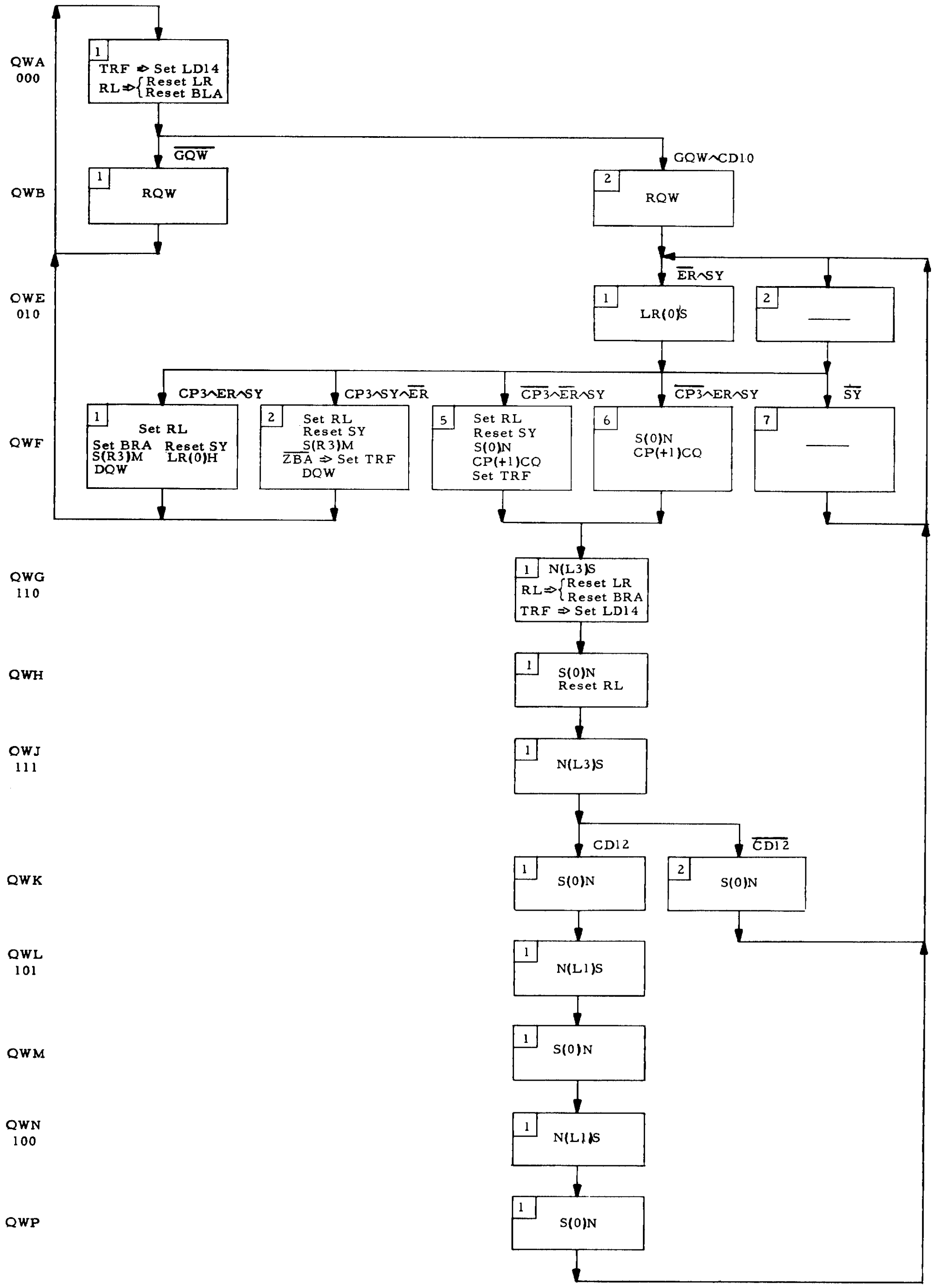


FIGURE 12.8–3 The QW Algorithm for Block Transmit

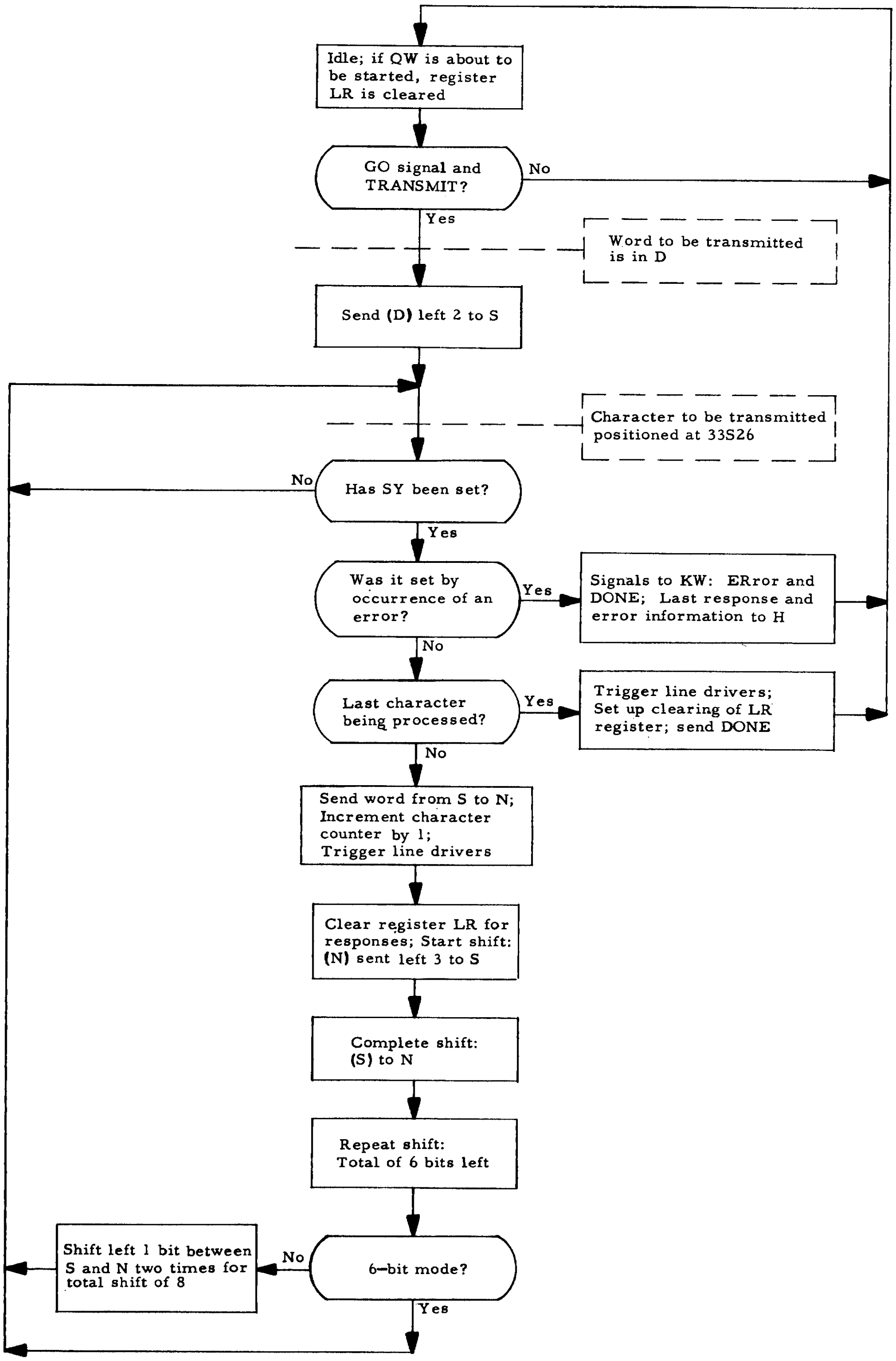

Idle; if QW is about to be started, register LR is cleared
GO signal and TRANSMIT?
No
Yes
Word to be transmitted is in D
Send (D) left 2 to S
Character to be transmitted positioned at 33S26
No
Has SY been set?
Yes
Was it set by occurrence of an error?
Yes
Signals to KW: ERror and DONE; Last response and error information to H
No
Last character being processed?
Yes
Trigger line drivers; Set up clearing of LR register; send DONE
No
Send word from S to N; Increment character counter by 1; Trigger line drivers
Clear register LR for responses; Start shift: (N) sent left 3 to S
Complete shift: (S) to N
Repeat shift: Total of 6 bits left
Shift left 1 bit between S and N two times for total shift of 8
No
6–bit mode?
Yes

TABLE 12.8–2 Terms Used on the QW Flow Chart for Block Transmit

(Most of these have been discussed elsewhere; short definitions are included at this point for ease of reference.)

BLA	Block Line receiver Amplifiers flip–flop.
BRA	BRAnch flip–flop.
CD10	Command decoding bit used to distinguish between transmit and receive operations; CD10 low indicates transmit.
CD12	Command decoding bit used to distinguish between 6 and 8–bit mode; CD12 high indicates 8–bit.
CP(+1)CQ	Increment path for CP–CQ character counter.
CP3	Signal sent by CP–CQ counter when a full word has been handled.
DQW	DONE QW sequencer.
ER	ERror signal.
LR(0)H	Path enabled when ER is high to send last response to the H register.
LR(0)S	Path enabled to send the last response to the least significant bits of the S register.
RL	Reset Line receiver register.
RQW	READY QW sequencer.
SY	SYnchronization flip–flops.

FIGURE 12.8–4 The QW Flow Chart for Block Transmit

QW

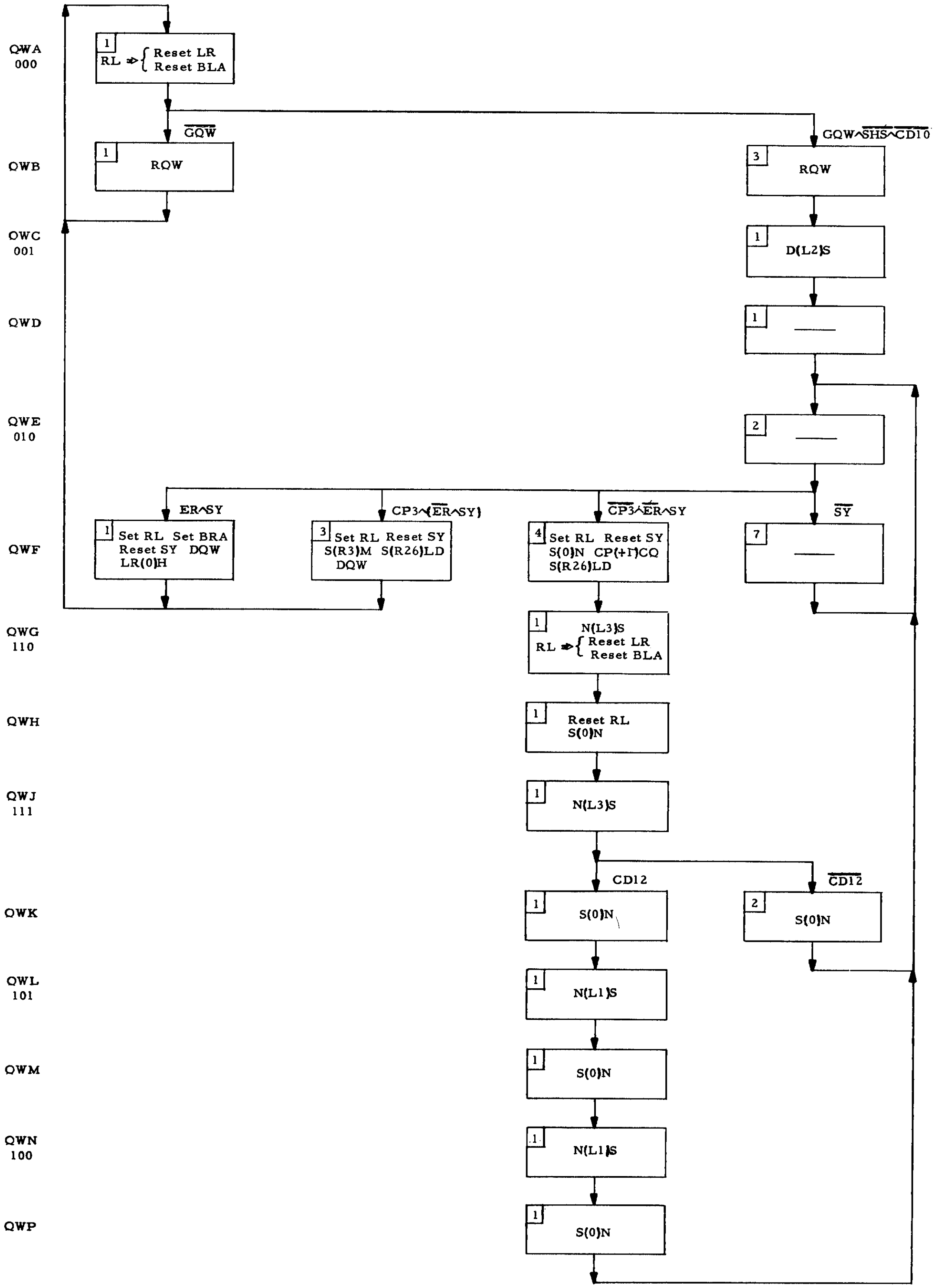
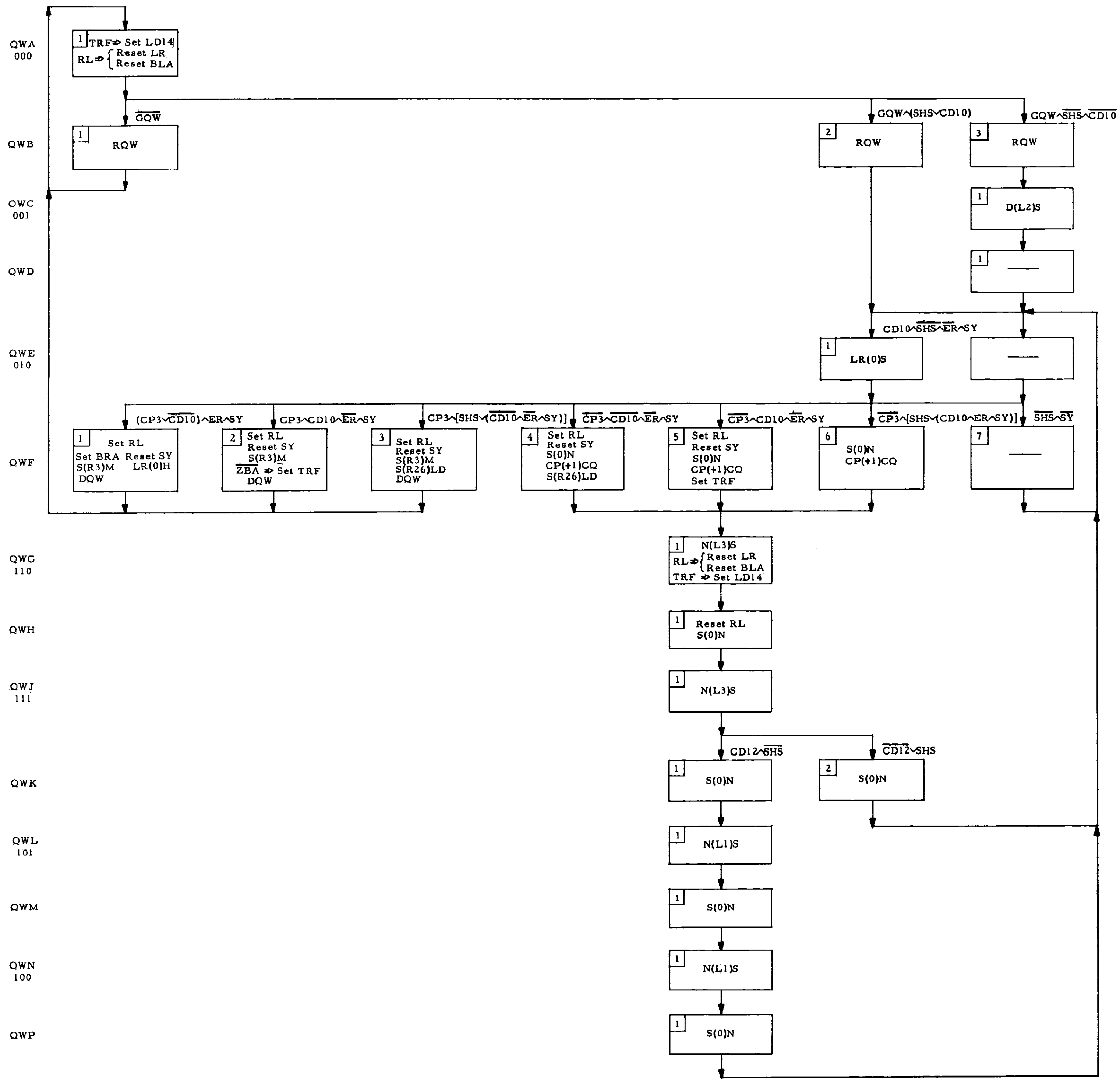

QWA 000
1 RL ⇒ { Reset LR Reset BLA
$\overline{GQW}$
$GQW \wedge \overline{SHS} \wedge \overline{CD10}$
QWB
1 RQW
3 RQW
QWC 001
1 D(L2)S
QWD
1 ——
QWE 010
2 ——
$ER \wedge SY$
$CP3 \wedge (\overline{ER} \wedge SY)$
$\overline{CP3} \wedge \overline{ER} \wedge SY$
$\overline{SY}$
QWF
1 Set RL Set BRA Reset SY DQW LR(0)H
3 Set RL Reset SY S(R3)M S(R26)LD DQW
4 Set RL Reset SY S(0)N CP(+Γ)CQ S(R26)LD
7 ——
QWG 110
1 N(L3)S RL ⇒ { Reset LR Reset BLA
QWH
1 Reset RL S(0)N
QWJ 111
1 N(L3)S
CD12
$\overline{CD12}$
QWK
1 S(0)N
2 S(0)N
QWL 101
1 N(L1)S
QWM
1 S(0)N
QWN 100
1 N(L1)S
QWP
1 S(0)N

FIGURE 12.8–5 The QW Flow Chart

QW


QWA 000
QWB
QWC 001
QWD
QWE 010
QWF
QWG 110
QWH
QWJ 111
QWK
QWL 101
QWM
QWN 100
QWP
1 TRF ⇒ Set LD14
RL ⇒ { Reset LR Reset BLA
$\overline{GQW}$
1 RQW
GQW∧(SHS∨CD10)
2 RQW
GQW∧$\overline{SHS}$∧$\overline{CD10}$
3 RQW
1 D(L2)S
1 ——
CD10∧$\overline{SHS}$∧$\overline{ER}$∧SY
1 LR(0)S
——
(CP3∨$\overline{CD10}$)∧ER∧SY
1 Set RL Set BRA Reset SY S(R3)M LR(0)H DQW
CP3∧CD10∧$\overline{ER}$∧SY
2 Set RL Reset SY S(R3)M $\overline{ZBA}$ ⇒ Set TRF DQW
CP3∧[SHS∨($\overline{CD10}$∧$\overline{ER}$∧SY)]
3 Set RL Reset SY S(R3)M S(R26)LD DQW
$\overline{CP3}$∧$\overline{CD10}$∧$\overline{ER}$∧SY
4 Set RL Reset SY S(0)N CP(+1)CQ S(R26)LD
$\overline{CP3}$∧CD10∧$\overline{ER}$∧SY
5 Set RL Reset SY S(0)N CP(+1)CQ Set TRF
$\overline{CP3}$∧[SHS∨(CD10∧ER∧SY)]
6 S(0)N CP(+1)CQ
$\overline{SHS}$∧$\overline{SY}$
7 ——
1 N(L3)S RL ⇒ { Reset LR Reset BLA TRF ⇒ Set LD14
1 Reset RL S(0)N
1 N(L3)S
CD12∧$\overline{SHS}$
1 S(0)N
$\overline{CD12}$∨SHS
2 S(0)N
1 N(L1)S
1 S(0)N
1 N(L1)S
1 S(0)N

CHAPTER 13

ADDER SEQUENCERS

SECTION 13.1 – INTRODUCTION

The sequencers that are described in this chapter process opcodes that call for use of the Adder. The operation, design, and cycle time of the Adder are discussed in Section 5.3. The speed of the add cycle is such that the result is always available by the time the logic calls for it. This speed is an important factor in the processing times for all arithmetic and logic operations, and particularly for the multiply and divide opcodes which use the Adder several times during the processing of a single command (in the formation of the remainder or of the partial product). Processing times are also affected by the basic simplicity of binary arithmetic. This is, of course, a factor in the speed of the Adder. It is also important in the multiply and divide algorithms where a significant amount of processing time is saved due to the fact that multiples of the divisor and the multiplicand are never involved in the calculation. The divisor can either be successfully subtracted from the numerator or not so that determination of the quotient bit becomes a simple branch: 1 or 0. Similarly, the multiplicand is added to the partial product or not depending upon whether the multiplier bit is 1 or 0. Applying the principles of non-restorative divide and string multiply to the basic divide and multiply algorithms, respectively, results in very fast processing times for these opcodes.

The Adder has two modes of operation: if the LP flip-flop (Logical Product) is set, the Adder forms the logical product of (N) and (D); if LP is reset, it forms the arithmetic sum. This is the only adaptation possible in the operation of the Adder. All other aspects of the particular operation must be handled by the controlling sequencers which set up the operands in the desired format in the N and D registers and set or reset the LP flip-flop. Exponent and sign values of the operands are ignored by the Adder; evaluation of these factors is left entirely to the sequencer logic.

The logic mode of the Adder is not used in the processing of all logic commands. (The KL and KJ sequencers, which handle almost all of the logic commands, set LP only for the R2, R3, L4, L5, S4, and S5 opcodes.) For the remaining logic operations, processing is carried out with LP reset. The operands are set up so that gating of the sum from the Adder yields the desired result (see Section 13.5). Logic operations require distinct formatting: the operands are shifted to zero exponent and truncated to 32 bits prior to being sent through the Adder. (This does not include commands L2, L3, S2 and S3 written in mode 0 or 1 which are handled in number format.) In addition to this, sign manipulations, which figure importantly in the processing of arithmetic operations, do not affect the processing of logic commands since logic words are unsigned.

Sign and exponent values are of great importance in the processing of arithmetic operations. The KC, KA, and KJ sequencers have similar requirements with respect to these factors and, thus, use a common Q level sequencer, QS, for the formation of the sum or difference. The multiply and divide algorithms have distinct needs and do not, therefore, use QS but rather gate the sum or difference directly from the Adder. Little need be said at this point concerning the handling of exponents: users of QS set up the operands in the N and D registers with the respective exponents in EA and EP; QS carries out exponent equalization, if necessary, before proceeding with the sum or difference operation. Handling of exponents in multiply and divide operations involves several sequencers; this is described in detail in Sections 13.8 through 13.12. It is appropriate, however, to discuss sign manipulations at this point since, even for the logic opcodes in which the operands are unsigned, there is a time during the operand assembly process at which the operands have attached signs, and, for arithmetic opcodes, the development of the final sign starts at operand assembly and continues to the end of the processing. A general discussion should forestall unnecessary repetitions each time a sign value is changed.

Sign values are stored in the AS, WS, and SM flip–flops and in the 29th bit of signed numbers in memory (integers, single and double precision numbers). This latter sign value is read from or stored into the 29th bit of register B when the word is being READ or WRITTEN. Hence, it is referred to as the B28 flip–flop. For the AS, WS, and B28 flip–flops, the low state indicates a positive sign; for the SM flip–flop, the low state indicates a negative sign. The SM flip–flop is used to indicate whether a summing or differencing operation is called for (SM is high for summing). The other sign flip–flops are associated with operands as follows:

1) flip–flop AS holds the sign of the operand stored in the Accumulator;
2) flip–flop WS holds the sign of the operand stored in OA.

Some general statements can be made concerning the status of each flip–flop at certain stages of Master Control.

1) At KCA just prior to command access: the SM flip–flop is set, the WS flip–flop reset to indicate a cleared condition for

these flip–flops. The exceptions to this are the OA commands (N0–7 and R2) which leave useful information in OA and, therefore, the sign value of this information in SM and WS; flip–flop B28 holds bit 28 of the last word read from or written into memory; the AS flip–flop holds the sign of the operand stored in the Accumulator.

2) At KCH, just following command access: the B28 flip–flop holds mode information for the command; the SM, WS, and AS flip–flops are unchanged.

3) At KCM, when the accessed information is available in register B: the B28 flip–flop holds the sign of the accessed information (unless this is a logic word accessed in logic format: an operand of a logic repeat operation or the final access of the operand for a mode 2 or 3 logic command in which case it holds bit 28 of the logic word); the WS and SM2 flip–flops hold the sign of the partially assembled operand or, if this is the first step in operand assembly, both hold a positive sign; (if B28 is high on a number access, the state of SM1 is being reversed); the AS flip–flop is unchanged.

4) At opcode startup: the sign of the assembled operand is in the WS and SM flip–flops; the AS flip–flop is unchanged; the B28 flip–flop reflects the sign of the last information accessed. At opcode startup the state of the WS flip–flop is checked for all opcodes which call for use of the assembled operand as an address. If WS is high (indicating negative) and the address is non–zero, an interrupt is requested and opcode startup bypassed.

From the time the K level sequencers are started, sign values are important only in handling of arithmetic and logic operations. The A, D, and L commands call for storing of the result in the Accumulator.

This means that the sign of the final value will be copied into the AS flip–flop. In the case of logic commands, the sign will be that of the result only in the following cases:

1) L2 and L3: the logic commands calling for arithmetic operations and, therefore, processing by KA; (result is shifted to zero exponent and truncated to 32 bits);

2) L0 and L1 written in modes 0 or 1: these commands call for the use of only one operand; in modes 0 and 1, logic command accesses are in number format; hence, the sign saved will be that of the operand accessed. (For the L0 and L1 commands in modes 2 and 3, the sign is made positive.)

Otherwise, for logic commands, the state of the AS flip–flop is unchanged by the operation and thus retains the sign of the last operand stored in the Accumulator. For S and T test commands, the contents of the Accumulator are unchanged unless the command is a repeat command in which case the Accumulator will contain the address plus one of the last operand processed and the AS flip–flop is accordingly made positive.

For the multiply/divide opcodes, determination of the final sign is a simple operation performed by the QA sequencer. If the sign of the operand stored in the Accumulator is positive, the sign of the result is that of the operand in OA (the sign held in the WS flip–flop); if the Accumulator sign is negative, the sign in WS is reversed and, thus, becomes the final sign. This sign is copied into the AS flip–flop as the final sign value. It should be pointed out that other pseudo sign manipulations are carried on during the processing of these commands. These in no way affect the true sign of the result which is already stored in the AS flip–flop; they serve only as temporary reminders of the size of the current remainder or partial product (see Sections 13.8 through 13.12).

The remaining arithmetic operations call for the determination of a sum or difference of two operands under control of the QS sequencer. When QS is used, the controlling sequencer sets up the operands in the N and D registers, with exponents in EA and EP and signs in WS and SM, and sends a GO signal to QS. QS handles exponent equalization, complements one of the operands for differencing operations, determines the final sign for differencing operations, gates the result to the S register, transfers it to N, and sends a DONE signal. QS is started by KC during operand assembly, by KA for each opcode handled by that sequencer, and by KJ for execution of the X2 (SKP) opcode. The QS inputs are shown in Table 13.1–1.

Following command access, QS is started one or more times during operand assembly. If the command is one of those calling for use of QS during processing by KA, the operand is transferred to registers D and EP while the second operand, if any, is transferred from the Accumulator to OA, and QS is started again. For the repeat arithmetic commands that use the Adder, QS is started for each iteration. Sign manipulations during this process seem to be enormously involved unless the basic actions of QS are understood.

Three sign values are involved in these calculations: that of the operation called for (add or subtract) and that of each operand. QS has cognizance of only two sign values: those held in SM and WS. At GQS, the state of SM reflects the combination of the three signs mentioned above. SM high indicates that a summing operation is called for (addition of two positive or two negative values); SM low indicates a differencing operation. The WS flip–flop holds the sign of the operand stored in register N. For summing operations, where both operands have the same sign, the sign in WS is the sign of the result. For differencing operations, this is the final sign only if the larger operand is stored in register N. Otherwise, the state of WS is

TABLE 13.1-1 QS Inputs

Sequencer	OA: Registers N and EA, Flip-Flop WS	Registers D and EP, Flip-Flop SM
KC (operand assembly)	Partially assembled operand with exponent and sign or, at start of operand assembly, zeros with zero exponent and positive sign.	Mantissa and exponent just accessed (if logic format, zero exponent); SM reflects sign held in WS (positive if at start of operand assembly); if B28 high (negative sign on accessed information), SM state is reversed (SM made positive if logic access).
KJ	Assembled operand with zero exponent; WS contains sign.	Contents of register CA (Next command address); zero exponent; SM reflects sign of WS.
KA	Operand from Accumulator; registers A and AE, flip-flop AS.	Assembled operand transferred from OA.

reversed to obtain the final sign. At DONE QS, the WS and SM flip-flops both hold the sign of the result so that SM is already taking into account one of the values necessary in determination of the next sign value. (Recall that SM high holds the same sign value as WS low; thus, at DQS, the flip-flop states are either $\overline{\text{WS}}$ and SM or WS and $\overline{\text{SM}}$.)

One of the inputs to QS is always the assembled operand which, at opcode startup, is stored in registers N and EA with the sign value held in SM and WS as described above. During operand assembly

(KC sequencer) the partially assembled operand in OA is added to the accessed information which will be sent to the D and EP registers. If the accessed operand is negative (B28 high), the state of the SM flip–flop is reversed. In KA's execution of arithmetic operations involving two operands, the second operand is in the Accumulator registers A and AE, with sign value in AS. KA transfers the assembled operand to D and EP and the Accumulator operand to N and EA. The sign in AS is sent to WS (WS is assumed to represent the sign of the operand held in N). The SM flip–flop is adjusted to represent the combination of the three sign values (see table below). The KJ sequencer inputs to QS are the assembled operand and the next command address from the CA register. This address is sent to register D and, since addresses have zero exponents, the EP register is cleared. KJ is adding the assembled operand to the next command address. The sign of the address is understood to be positive so that the state of SM (which reflects the sign of the assembled operand) is not changed before QS is started. If the operand is negative, SM low will correctly indicate a differencing operation.

TABLE 13.1–2 Sign Determination

Sign held in WS and SM following last operation of QS:	+	+	−	−	+	+	−	−
Sign of second operand (Held in AS or B28):	+	−	+	−	+	−	+	−
Sign of operation (Add or subtract):	+	+	+	+	−	−	−	−
Sign in SM at GO QS:	+	−	−	+	−	+	+	−
Sign in WS and SM At DONE QS:	+	?	?	−	?	+	−	?

The preceding table summarizes the factors involved in determination of the final sign at DONE QS.

The question marks indicate that the final value is that of the larger operand. Notice that the manipulations are reversed for the add and subtract operations.

Table 13.1–3 relates the various sequencer operations and the way in which the Adder is used to carry them out; Table 13.1–4 identifies the princiapl signals used or generated by the Adder circuitry.

TABLE 13.1–3 Comparison Between Sequencers That Use the Adder

Adder users: (Initialization and overall control)	KC	KA	KJ	KL	KJ	KD	KD
Adder used to form:	Sum/Difference			Logical Product or Logical Sum	Logical Product	Partial Product	Remainder
Output used as:	Current value of assembled operand	Final result of current arith-metic operation	Next command address	Final value of current logical operation	Means of reading contents of designated register	Current partial product	Current remainder value
Adder input from register N:	Partially assembled operand	Contents of Accumulator	Assembled operand	Contents of Accumulator	Assembled operand (extractor)	Multiplicand (binary normalized)	Remainder (Numerator binary normalized)
Adder input from register D:	Information just accessed	Assembled operand	Contents of CA register	Assembled operand	Contents of register to be extracted	Partial Product	Complemented denominator (binary normalized)
LP Flip-flop:	Reset	Reset	Reset	Set to form logical product, reset to form logical sum	Set	Reset	Reset
Output from Adder gated by:	QS	QS	QS	KL	KJ	QP	QQ
Exponent manipulations:	QS equalizes exponents prior to add cycle, thus forming final exponent			Operands have zero exponents	Operands have zero exponents	Formation of final exponent involves QA, KD and QP	Formation of final exponent involves QA, KD and QQ
Enabled output path:	R(0)S	R(0)S	R(0)S	R(L1)S	R(0)S	R(0)S R(L1)S	R(0)S R(L1)S
Result sent from S to N; at opcode completion, stored in:	N, exponent in EA, sign in WS	A, exponent in AE, sign in AS	Register CA	A, exponent in AE, sign in AS	A if ERA (R3) N if ERO (R2)	A, exponent in AE, sign in AS	A, exponent in AE, sign in AS

N.B. KA, KC, and KD do not use the adder directly; rather, they control operations that involve use of the adder. KJ uses the adder directly in two instances, indirectly in one. The relationships that exist between the controlling sequencers and the adder are shown above.

TABLE 13.1–4 Signals Important in Operation of the Adder

CY	Carry signal CY is generated if there is a mantissa overflow from the Adder and the carry out flip–flop (CYE) is set.
CZ	Carry Zero flip–flop; CZ is set to cause a carry in to the zero bit of the Adder.
DB	Discarded Bits flip–flop; DB is set when any of the 3 bits shifted out of register D on a right 3 shift is a 1; thus $DB \Rightarrow D{-}1 \vee D{-}2 \vee D{-}3$. This information is used in evaluating the state of the CZ flip–flop.
LFC, LFK, PC, PK	LeapFrog Carry, LeapFrog Kill, Propagate Carry, Propagate Kill; signals used by the Adder in determination of the result; see Section 5.3 for a description of the use of these terms.
LP	Logical Product flip–flop; when LP is set the Adder output is the logical product or extraction of the two operands.
SM	Sum flip–flop; SM is high to indicate a summing operation in the QS sequencer; $\overline{SM}$ indicates a differencing operation. In the operation of QS, SM sets CZ in anticipation of an end–around carry (see Section 13.6).
S42	Bit 42 in register S; if the Adder output is gated R(0)S, S42 high sets CY if CY is enabled.
S43	Bit 43 in register S; if the Adder output is gated R(L1)S, S43 high sets CY if CY is enabled.
R(0)S, R(L1)S	Paths that can be enabled to gate output from the Adder.
RU	Round–Up flip–flop; RU is set when the number held in the S register has as its least significant octal digit a 5, 6 or 7, or if it has a 4 in conjunction with either the DB signal high or the second least significant octal digit is an even number; thus, $RU \Rightarrow S2 \wedge (S0 \vee S1 \vee \overline{S3} \vee DB)$. This produces a non–biased round–up.

SECTION 13.2 – REPEAT OPERATIONS

The command format used to designate repeat operations is basically the same as that described in the chapter on block input/output. The assembled operand of the first command word is the address of the first location in the block; the opcode for repeat is 013 (input/output is 033). Master Control checks the operand for address format and sends a GO signal to KM. The second command word, accessed by QC, contains less information than does the second word in block input/output, but no distinction is made between the two cases by QC. (Refer to Section 12.6 for a discussion of the QC sequencer.) The second word contains the block length and the opcode that is to be repeated. The difference between the two lies in the lack of any information in bits 22–15 for repeat commands. (In block input/output, these bits are used to specify the starting instruction. Transmission of this instruction to the instructed device is carried out by the input/output sequencers.) For repeat commands, the only information required in this word is the block length and designation of the opcode that is to be repeated.

Of the opcodes that call for use of the Adder, 32 can be repeated: A, T, L, and S. Thus, the KA and KL sequencers which handle these commands operate in the repeat and non–repeat modes. KA can be started by either KC or KM; KL is started by KC in all cases. Recall that two of the logic operations and two of the logic tests are handled by KA, the arithmetic sequencer, rather than KL because they call for performance of arithmetic operations. This leaves 12 opcodes for handling by KL in the repeat and non–repeat modes. KA processes 26 opcodes in non–repeat and 20 in repeat. (The additional 6 in non–repeat are the address preparation opcodes N2–7 which are not available in the repeat mode.) In order to understand the functioning of the KA and KL sequencers in both repeat and non–repeat modes, it is necessary to be aware of the basic simplicity of the opcode list with regard to the arithmetic and logic commands. There is no distinction between the processing of the N and A commands except that the N commands call for storage of the result in OA, while A commands store them in the Accumulator. The arithemtic test commands also call for performance of the same operation with the additional requirement that the result obtained be tested. This correspondence between the A and T commands is similar to that which exists between the L and S commands.

The value of the T and S test commands is that they provide the programmer with a means for causing a branch in the program when a certain condition is met. For the T and S2, S3 commands, the test is satisfied when the result is positive and non–zero; the remaining S commands are satisifed when the result is zero. Following each test command, the programmer uses two words to stipulate the action to be taken next. The next command is a transfer command that is used to cause a branch to some other part of the program. This command is executed when the test conditions are satisfied. The next command plus one is taken when the test conditions are not satisfied. In programming jargon, the case where the test is satisfied and the next command taken is known as the transfer branch because the command used in that location is normally a transfer command. This should not be confused with the use of the term jump to describe the incrementing of the next command address when the test conditions are not satisfied. Jump simply refers to skipping the next command.

The similarities between groups of opcodes can be analyzed further. For the N, A and T commands numbered 2–7, the even–numbered operate on the quantity ((Acc) + **X**), the odd–numbered on ((Acc) – **X**). This rule also applies to the logic commands L2, L3, S2 and S3. Further, the N, A and T commands numbered 2–7 can be grouped as follows:

1) Those numbered 2 or 3 give a positive sign to the result; for the test commands, the test is satisfied if the result is positive and non–zero; when satisfied, a transfer in the program occurs via the next command.

2) Those numbered 4 and 5 give a negative sign to the result; for the test commands, the test is satisfied if the result of the negation is positive; when satisfied, a transfer in the program occurs via the next command.

3) Those numbered 6 and 7 take the absolute value of the result; for the test commands, the test is satisfied if the absolute value of the result is greater than zero; when satisfied, a transfer in the program occurs via the next command. (This applies also to the S2 and S3 test commands.)

In the processing of the L and S commands (except L2, L3, S2 and S3) the logical analog of the positive–negative rule is used. The even–numbered opcodes operate on ((Acc) $ X), the odd–numbered on ((Acc) $ $\overline{X}$) where $ indicates a logical operation and $\overline{X}$ indicates the complement of X. Logic opcodes 4 and 5 call for the formation of the logical product, while 6 and 7 call for logical sum. Logic tests are satisfied when the resulting quantity equals zero.

The single operand commands, numbered 0 and 1, involve the quantity +X or –X for the N, T and A commands, and X or $\overline{X}$ for the S and L commands.

From a programming point of view, these commands afford great variety; from the point of view of the logic used to implement these commands, they are not very different from one another. Versatility is gained through the manipulation of signs, the placement or the formatting of the result, and, above all, the interpretation of what has taken place. To demonstrate the flexibility obtained through sign manipulations, consider the handling of the N2–N7 commands (address preparation opcodes) by KA. When QS has summed or differenced the two operands, processing is handled as follows:

1) N2 and N3 call for storing the result from the sum or difference operation in OA (N, EA, WS). Since QS leaves the result in OA, no further action is required. Thus, control is returned to KC;

2) The N4 and N5 commands call for the negation of the result to be left in OA; thus, the sign in WS is reversed and control is returned to KC;

3) The N6 and N7 commands call for the absolute value of the result to be left in OA; thus, the WS flip–flop is reset; control is returned to KC.

To demonstrate the simplicity involved in the processing of test commands, consider the test T4 (iF Sum Minus, FSM) which calls for a transfer if $(Acc) + X < 0$. For programming purposes, a negative result satisfies this test and calls for execution of the next command. However, the logic in KA says that the test is satisfied when the result is positive and non–zero. Hence, the processing of the command involves reversing the meaning of the result of the arithmetic operation as follows:

1) QS performs the summing of (Acc) + X;

2) The sign of the result, held in WS, is complemented. This means that a case that satisfies the criterion and does, in fact, show negative sign, will become positive before the test occurs and thus also satisfy the logic. The reverse is obviously true also;

3) The result is sent to the S register to be tested against zero;

4) If the result is either zero or negative, the test fails and the address of the next command is incremented by one to cause a jump in the program.

Similar manipulations adapt all tests to the non–negative, non–zero requirement. The logic tests handled by KA are not concerned with signs since there can be no such thing as a negative logic word; thus, these are gated through this part of the sequencer on free positive signs so that they will not be mistakenly held up.

The processing of logic opcodes by KL differs in three respects from that carried out for arithmetic opcodes by KA. These distinctions are shown in Table 13.2–1.

Despite the distinctiveness of these operations, the basic manipulations still closely resemble those involved in the processing of arithmetic operations. The commands involving a single operand – L0, L1, S0, S1 – comply with the rules mentioned above but are otherwise handled no differently than the single operand arithmetic commands. The commands L4, L5, S4 and S5 require determination of the logical product. This process, also referred to as ANDing, calls for 1 bit in the result wherever there is a 1 bit in both operands. The result is obtained by the Adder operating with the LP flip–flop set. The opcodes numbered 6 and 7 call for uniting (ORing) the two operands. This operation results in a 1 bit wherever a 1 bit exists in either operand. To perform the union, one operand is stored in S, the other in D, and N is cleared. The Adder adds (D) to (N), that is, to 0, and transfers the result to S. Since the transfer into S is one–sided, the result of superimposing (D) onto (S) is the union of the two operands.

TABLE 13.2–1 Processing Distinctions Between KA and KL

Sequencer:	KA	KL
Operand Format:	Logic format for L2, L3, S2 or S3 commands written in mode 2 or 3; otherwise, number format.	Logic format (zero exponent with truncation to 32 bits).
Quantity involved in even–numbered opcodes:	((Acc) + X)	((Acc) \$ X) (\$ = logic operation)
Odd–numbered opcodes:	((Acc) – X)	((Acc) \$ $\bar{X}$)
Test commands satisfied if:	Result positive	Result zero

Logic tests handled by KL are processed so that they are satisfied when the result is zero. For example, the command S5 (If Extraction of Complement zero) is handled as follows:

1) both operands are put in logic format;
2) the operand from register N is complemented;
3) the logical product is gated from the Adder;
4) the result is sent to S for the zero test;
5) if the result is zero, computation will continue with the next command which will call for a branch in the program; if not zero, the next command address is incremented by one.

All of the commands handled by KA and KL, used individually, are valuable to the programmer. Not all of them are necessary in the repeat mode. In fact, the address preparation opcodes have been omitted from the repeat cycle, and some of those remaining are not really helpful. (There is little to be gained by clearing the Accumulator and adding the next operand over and over, but this is possible through use of the AO command.) However, repeat has been set up in such a way that it requires no additional logic to make all the A commands or all the L commands available in this mode if some of them are valuable and should be included. Thus, the retention of repeat AO.

TABLE 13.2–2 Processing of the Arithmetic Test and Logic Test Commands

COMMANDS:	Non–Repeat AO–7	Repeat AO–7	Non–Repeat LO–7	Repeat LO–7
OPERAND IN N AT OPCODE STARTUP:	Double or single precision number, accessed in number format.	Block can contain mixed double and single precision numbers; each is accessed in number format.	MODE 0 or 1 command: single or double precision number, accessed in number format. MODE 2 or 3 command: single word accessed in logic format.	Block can contain only single words that will be accessed in logic format.
OTHER OPERAND IF ANY:	Contents of Accumulator	Contents of Accumulator which, after the first iteration, will be the result of the preceding operation.	Contents of the Accumulator	Contents of the Accumulator which, after the first iteration, will be the result of the preceding operation.
PROCESSING PROCEDURE:	Arithmetic operation performed; result (in number format) stored in the Accumulator.	Arithmetic operation performed; storage of result in Accumulator makes it available as one of the operands for the next iteration. KA signals KC to access next operand in the block.	Contents of Accumulator shifted to zero exponent, truncated to 32 bits; logical operation performed; Result stored in Accumulator	**Storage of result in Accumulator makes it available as one of the operands for the next iteration; KL (KA if L2 or L3) signals KC to access next operand.**
CONTENTS OF ACCUMULATOR AT CONCLUSION:	Result of single operation	Result at time of termination	Result of single operation	Result at time of termination
REASONS FOR TERMINATION:		Enabled data flag, end of block, exponent overflow or memory overflow		Enabled logic flag, end of block, or memory overflow.

Some of the repeat commands are extremely important in providing programming flexibility. For example, if it is necessary in the execution of a program to examine a table of information in order to

TABLE 13.2–3 Processing of the Arithmetic Test and Logic Test Commands

	Handled by KA				Handled by KL	
	Non–Repeat		Repeat		Non–Repeat	Repeat
COMMANDS:	T0–7	S2, S3	T0–7	S2, S3	S0, S1, S4–7	S0, S1, S4–7
OPERAND IN N AT OPCODE STARTUP:	Double or single precision number, accessed in number format.	Same as that stated for S0, S1, S4–7	Block can contain mixed double and single precision numbers: each is accessed in number format.	Same as that started for S0, S1, S4–7	MODE 0 or 1 commands: single or double precision number, accessed in number format. MODE 2 or 3 command: single word accessed in logic format.	Block can contain only single words that will be accessed in logic format.
OTHER OPERAND IF ANY:	Contents of Accumulator		Contents of Accumulator		Contents of Accumulator	Contents of Accumulator
PROCESSING PROCEDURE:	Arithmetic operation performed; result sent to S for zero test; sign made positive for S2 and S3.		Same as for single operation with new operand being tested against contents of Accumulator at each iteration.		Contents of Accumulator shifted to zero exponent, truncated to 32 bits; logical operation performed.	

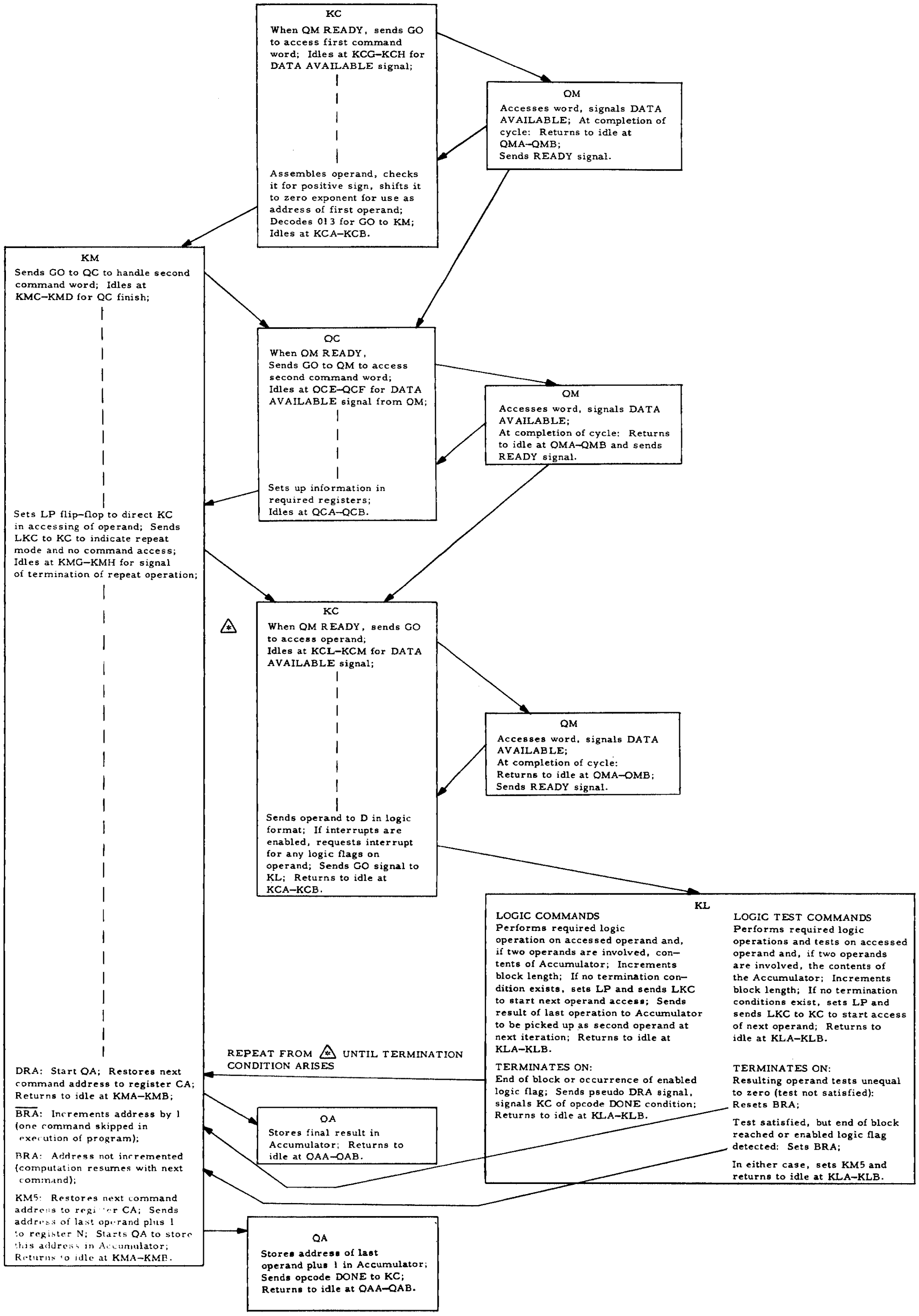
TEST BRANCHES:	TEST SATISFIED:
	Positive and non–zero–take next command.
	TEST NOT SATISFIED:
	Negative or zero–increment next command address by 1 for jump in program.
			IF TEST SATISFIED:
			Continue testing with next operand.
			IF TEST NOT SATISFIED:
			Repeat operation is terminated; increment next command address by 1 for jump in program.
					TEST SATISFIED:
					Result = 0 – take next command.
					TEST NOT SATISFIED:
					Result ≠ 0 – increment next command address by 1 for jump in program.
						IF TEST SATISFIED:
						Continue testing with next operand.
						IF TEST NOT SATISFIED:
						Repeat operation is terminated; increment next command address by 1 for jump in program.

TERMINATION DUE TO END OF BLOCK:			All tests satisfied; computation continues with next command.			All tests satisifed; computation continues with next command.
TERMINATION DUE TO INTERRUPT REQUEST SET BY ENABLED FLAG, MEMORY OVERFLOW, OR EXPONENT OVER–FLOW:			Next command or next command plus one taken depending upon the result of the last test before the interrupt was requested.			Next command or next command plus one taken depending upon the result of the last test before the interrupt was requested.
CONTENTS OF ACCUMULATOR AT TERMINATION:	Contents of Accumulator undisturbed by this operation.		Address plus one of last operand tested.		Contents of Accumulator undisturbed by this operation.	Address plus one of last operand tested.

determine whether or not it contains a particular word, the programmer can do so using the S3 command (IUO) in the repeat mode. If the word is in the table, the operation will be terminated on the operand and computation will continue with the next command. In addition to this, the address of the equivalent operand in the table, plus 1, is left in the Accumulator so that the programmer can determine the exact location of the word in the table. (This is only one of the possible ways to search the table; the programmer can choose the repeat command that best suits the situation.)

Since all of the opcodes discussed here are handled similarly, Tables 13.2–2 and 13.2–3 have been included in order to point up the distinctions between them. Figure 13.2–1 shows the tasks performed by the various sequencers involved in the repeat logic operations and tests that are handled by KL; Figure 13.2–2 gives the same information for repeat operations and tests handled by KA. (This includes the A and T commands as well as L2, L3, S2 and S3.) The control of repeat operations by the KM sequencer is described in Section 13.3.

FIGURE 13.2–1 General Algorithm for Processing Repeat Commands Handled by KL



KC

When QM READY, sends GO to access first command word; Idles at KCG–KCH for DATA AVAILABLE signal;

Assembles operand, checks it for positive sign, shifts it to zero exponent for use as address of first operand; Decodes 013 for GO to KM; Idles at KCA–KCB.

QM

Accesses word, signals DATA AVAILABLE; At completion of cycle: Returns to idle at QMA–QMB; Sends READY signal.

KM

Sends GO to QC to handle second command word; Idles at KMC–KMD for QC finish;

Sets LP flip–flop to direct KC in accessing of operand; Sends LKC to KC to indicate repeat mode and no command access; Idles at KMG–KMH for signal of termination of repeat operation;

DRA: Start QA; Restores next command address to register CA; Returns to idle at KMA–KMB;

$\overline{\text{BRA}}$: Increments address by 1 (one command skipped in execution of program);

BRA: Address not incremented (computation resumes with next command);

KM5: Restores next command address to register CA; Sends address of last operand plus 1 to register N; Starts QA to store this address in Accumulator; Returns to idle at KMA–KMB.

QC

When QM READY, Sends GO to QM to access second command word; Idles at QCE–QCF for DATA AVAILABLE signal from QM;

Sets up information in required registers; Idles at QCA–QCB.

QM

Accesses word, signals DATA AVAILABLE; At completion of cycle: Returns to idle at QMA–QMB and sends READY signal.

△

KC

When QM READY, sends GO to access operand; Idles at KCL–KCM for DATA AVAILABLE signal;

Sends operand to D in logic format; If interrupts are enabled, requests interrupt for any logic flags on operand; Sends GO signal to KL; Returns to idle at KCA–KCB.

QM

Accesses word, signals DATA AVAILABLE; At completion of cycle: Returns to idle at QMA–QMB; Sends READY signal.

KL

LOGIC COMMANDS

Performs required logic operation on accessed operand and, if two operands are involved, contents of Accumulator; Increments block length; If no termination condition exists, sets LP and sends LKC to start next operand access; Sends result of last operation to Accumulator to be picked up as second operand at next iteration; Returns to idle at KLA–KLB.

TERMINATES ON:

End of block or occurrence of enabled logic flag; Sends pseudo DRA signal, signals KC of opcode DONE condition; Returns to idle at KLA–KLB.

LOGIC TEST COMMANDS

Performs required logic operations and tests on accessed operand and, if two operands are involved, the contents of the Accumulator; Increments block length; If no termination conditions exist, sets LP and sends LKC to KC to start access of next operand; Returns to idle at KLA–KLB.

TERMINATES ON:

Resulting operand tests unequal to zero (test not satisfied): Resets BRA;

Test satisfied, but end of block reached or enabled logic flag detected: Sets BRA;

In either case, sets KM5 and returns to idle at KLA–KLB.

REPEAT FROM △ UNTIL TERMINATION CONDITION ARISES

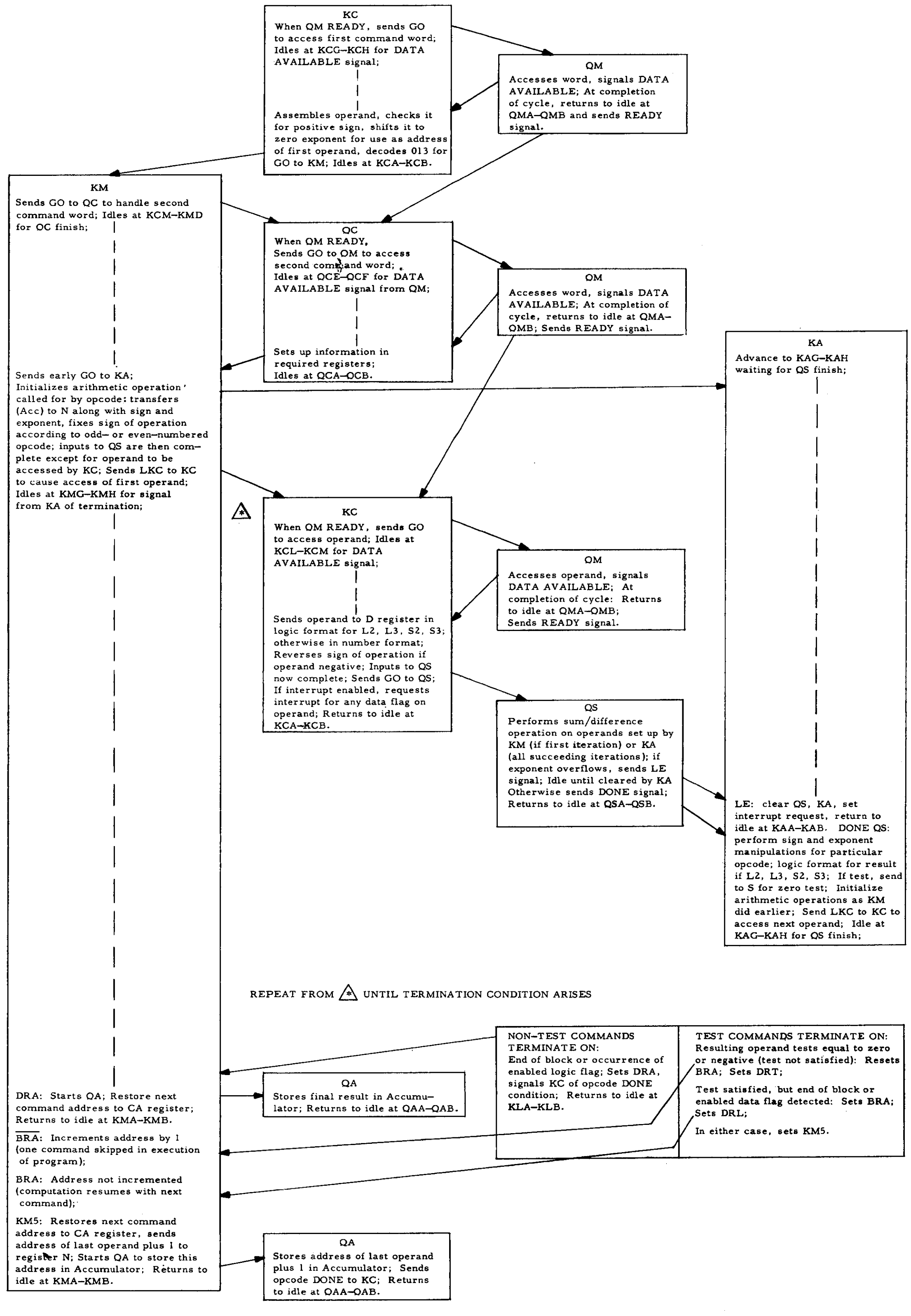
QA

Stores final result in Accumulator; Returns to idle at QAA–QAB.

QA

Stores address of last operand plus 1 in Accumulator; Sends opcode DONE to KC; Returns to idle at QAA–QAB.

FIGURE 13.2-2 General Algorithm for Processing Repeat Commands Handled by KA



KC
When QM READY, sends GO to access first command word; Idles at KCG-KCH for DATA AVAILABLE signal;

Assembles operand, checks it for positive sign, shifts it to zero exponent for use as address of first operand, decodes 013 for GO to KM; Idles at KCA-KCB.

QM
Accesses word, signals DATA AVAILABLE; At completion of cycle, returns to idle at QMA-QMB and sends READY signal.

KM
Sends GO to QC to handle second command word; Idles at KCM-KMD for OC finish;

Sends early GO to KA; Initializes arithmetic operation' called for by opcode: transfers (Acc) to N along with sign and exponent, fixes sign of operation according to odd- or even-numbered opcode; inputs to QS are then complete except for operand to be accessed by KC; Sends LKC to KC to cause access of first operand; Idles at KMG-KMH for signal from KA of termination;

DRA: Starts QA; Restore next command address to CA register; Returns to idle at KMA-KMB.

BRA: Increments address by 1 (one command skipped in execution of program);

BRA: Address not incremented (computation resumes with next command);

KM5: Restores next command address to CA register, sends address of last operand plus 1 to register N; Starts QA to store this address in Accumulator; Returns to idle at KMA-KMB.

QC
When QM READY, Sends GO to QM to access second command word; Idles at QCE-QCF for DATA AVAILABLE signal from QM;

Sets up information in required registers; Idles at QCA-QCB.

QM
Accesses word, signals DATA AVAILABLE; At completion of cycle, returns to idle at QMA-QMB; Sends READY signal.

KA
Advance to KAG-KAH waiting for QS finish;

LE: clear QS, KA, set interrupt request, return to idle at KAA-KAB. DONE QS: perform sign and exponent manipulations for particular opcode; logic format for result if L2, L3, S2, S3; If test, send to S for zero test; Initialize arithmetic operations as KM did earlier; Send LKC to KC to access next operand; Idle at KAG-KAH for QS finish;

△*

KC
When QM READY, sends GO to access operand; Idles at KCL-KCM for DATA AVAILABLE signal;

Sends operand to D register in logic format for L2, L3, S2, S3; otherwise in number format; Reverses sign of operation if operand negative; Inputs to QS now complete; Sends GO to QS; If interrupt enabled, requests interrupt for any data flag on operand; Returns to idle at KCA-KCB.

QM
Accesses operand, signals DATA AVAILABLE; At completion of cycle: Returns to idle at QMA-QMB; Sends READY signal.

QS
Performs sum/difference operation on operands set up by KM (if first iteration) or KA (all succeeding iterations); if exponent overflows, sends LE signal; Idle until cleared by KA Otherwise sends DONE signal; Returns to idle at QSA-QSB.

REPEAT FROM △* UNTIL TERMINATION CONDITION ARISES

NON-TEST COMMANDS TERMINATE ON:
End of block or occurrence of enabled logic flag; Sets DRA, signals KC of opcode DONE condition; Returns to idle at KLA-KLB.

TEST COMMANDS TERMINATE ON:
Resulting operand tests equal to zero or negative (test not satisfied): Resets BRA; Sets DRT;

Test satisfied, but end of block or enabled data flag detected: Sets BRA; Sets DRL;

In either case, sets KM5.

QA
Stores final result in Accumulator; Returns to idle at QAA-QAB.

QA
Stores address of last operand plus 1 in Accumulator; Sends opcode DONE to KC; Returns to idle at QAA-QAB.

SECTION 13.3 – KM SEQUENCER CONTROL OF REPEAT OPERATIONS

The operation of KM in the handling of repeat operations differs only slightly from that involved in the processing of block input/output. The QC sequencer is started to access the second word and set up the information contained therein. The operation of QC is the same for both repeat and block modes. When this is accomplished, KM sets up the first of the repeat operations, a process that will be referred to as initialization, and signals KC to access the first operand. If the repeated opcode is one of those processed by KA, a GKA signal is sent to advance KA to the KAG–KAH idle loop where it waits for a DONE QS signal. There are two reasons for doing this:

1) KC is designed so that a mode 2 or 3 command, or a repeat command, processed by KA can use the QS startup at KCM (or KCR for double precision accesses) to perform the arithmetic operation called for by the opcode. This is feasible because these accesses do not require operand assembly. In fact, the startup would be superfluous if not used in this way since it would be necessary to clear OA before sending the GQS signal and the accessed operand would then be added to zero. The handling of mode 2 or 3 commands destined for processing by KA was described in Chapter 9. KA transfers the second operand, if any, to OA, the accessed operand is sent to D, and a GKA signal advances KA. Following the transmission of the GQS signal, KC returns to normal idle. KA idles for the DONE QS signal and then proceeds with its normal operations. Similarly, KM sends the second operand, if any, to OA for the first iteration of repeat commands, and a GKA signal to KA. An LKC signal instructs KC to access the operand. Following transmission of the GQS signal, KC returns to the normal idle

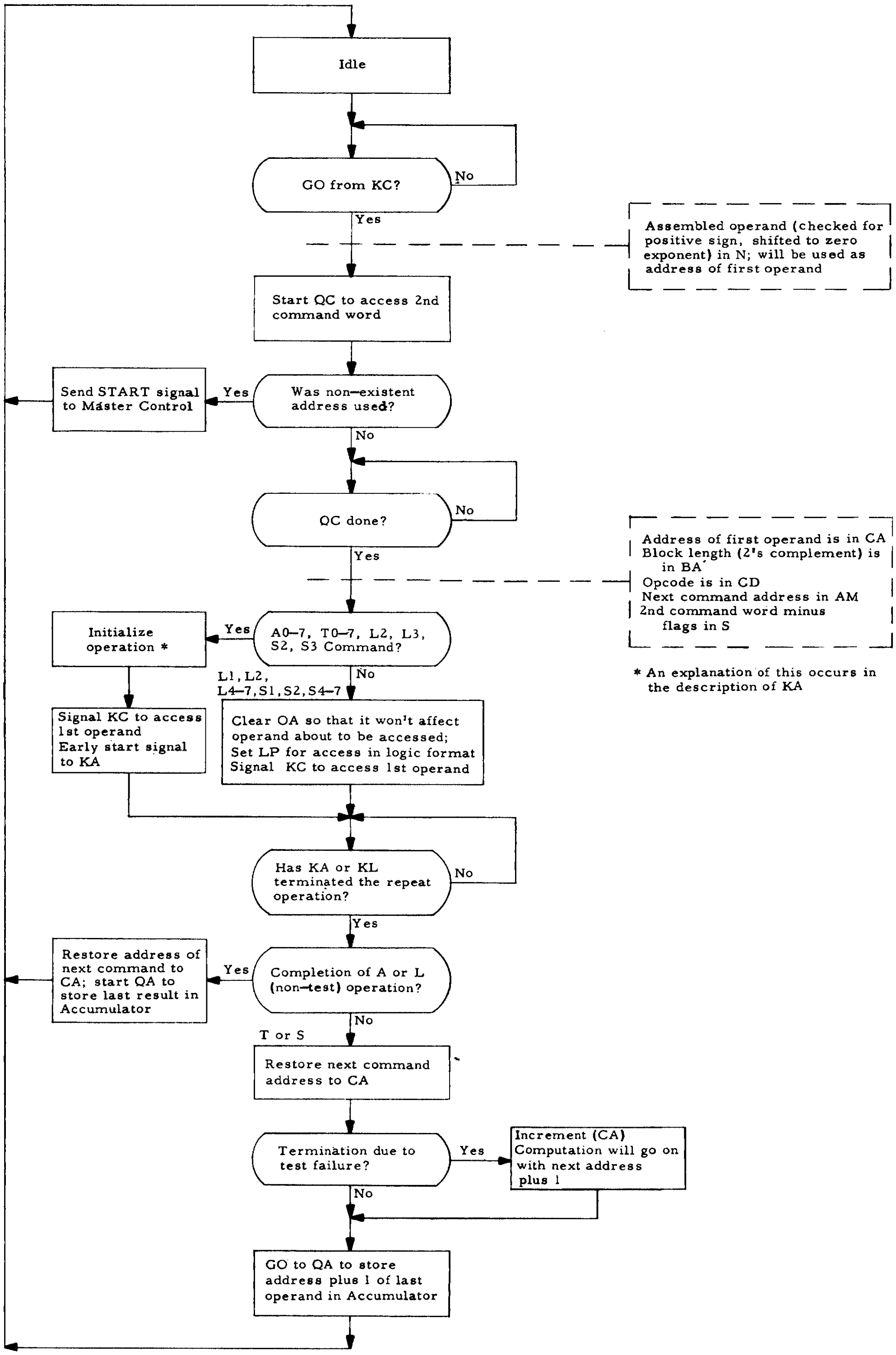
loop while KA waits for the DQS signal. On subsequent iterations, KA sends the second operand to OA, the LKC signal to KC, and goes automatically to the KAG–KAH idle loop.

2) The gating used by KC to send it back to its normal idle loop following the startup of QS rather than to opcode startup is the term KA6. KA6 is part of the KAG–KAH decoding and is high only if the KA sequencer is in operation. Thus the early advancement of the KA sequencer effects the necessary branch.

If the repeated opcode is one of those processed by KL, it is impossible for KM to speed up the operation because KL does not use the QS sequencer. KM does clear OA so that the accessed operand will not be affected by any residual information in that register when GQS is sent, and the LP flip–flop is set to gate an operand access in logic format and immediate startup of KL. (This is the path taken on the final operand access of mode 2 and 3 non–repeat opcodes in this group.)

The sign manipulations throughout KM are in accordance with the algorithm described in Section 13.1 and will not, therefore, be discussed in detail in this section. Figure 13.3–1 shows the algorithm for the KM control of repeat operations and Figure 13.3–2 contains the corresponding section of the KM flow chart. Figure 13.3–3 is the flow chart for the entire KM sequencer.

FIGURE 13.3—1 Algorithm for KM Control of Repeat Operations


Idle
GO from KC?
No
Yes
Assembled operand (checked for positive sign, shifted to zero exponent) in N; will be used as address of first operand
Start QC to access 2nd command word
Send START signal to Master Control
Yes
Was non—existent address used?
No
QC done?
No
Yes
Address of first operand is in CA
Block length (2's complement) is in BA
Opcode is in CD
Next command address in AM
2nd command word minus flags in S
Initialize operation *
Yes
A0—7, T0—7, L2, L3, S2, S3 Command?
L1, L2, L4—7, S1, S2, S4—7
No
Signal KC to access 1st operand
Early start signal to KA
Clear OA so that it won't affect operand about to be accessed;
Set LP for access in logic format
Signal KC to access 1st operand
Has KA or KL terminated the repeat operation?
No
Yes
Restore address of next command to CA; start QA to store last result in Accumulator
Yes
Completion of A or L (non—test) operation?
No
T or S
Restore next command address to CA
Termination due to test failure?
Yes
Increment (CA)
Computation will go on with next address plus 1
No
GO to QA to store address plus 1 of last operand in Accumulator

* An explanation of this occurs in the description of KA

TABLE 13.3–1 Terms Used on KM Flow Chart for Control of Repeat Operations

BRA	BRAnch flip–flop; BRA is set when a repeat test operation terminates due to end of block or enabled flag; program continues with the execution of the next command.
BU(0)D	Current information on the bus sent to register D.
Clear OA	Clear N, clear EA, reset WS, set SM.
DKM	DONE signal from KM sequencer.
DRA	Done Repeat Arithmetic operations (including L2 and L3); high when the end of the block is reached, an enabled flag detected or MNA goes high during the processing of these opcodes.
DRL	Done Repeat test signal; high when repeat test terminated due to end of block or enabled flag; program will continue with execution of the next command DRL => set BRA.
DRT	Done Repeat Test signal; DRT is high when the test is not satisfied; program will continue with execution of the next command plus 1.
EVEN	Even–numbered opcodes.
GKA	GO signal to KA sequencer.
GKC	Computed signal used to advance KC; GKC sets KR.
GKM	GO signal to KM sequencer for repeat operations.
GQA	GO signal to QA sequencer.
GQC	Go to QC sequencer; there is actually no such signal, QC is started when KM sets the QC1 flip–flop.
GRS	Go Repeat Sequencer flip–flops; GRS => LKC.
Group 2f	A0, A1, T0, T1.
Group 2h	A2–7, T2–7, L2, L3, S2, S3 (this is the group gated through KC by 2g, but without the N commands which do not occur in the repeat mode).
Group 4c	L0, L1, L4–7, S0, S1, S4–7.
KMR	KM sequencer Ready signal; KMR => DKM.
KR	Opcode DONE signal to KC.
LKC	Loop KC signal; LKC directs KC to access the next operand in a repeat operation.
LP	Logical Product flip–flop; LP is set at KMD to direct KC to access the first operand in the repeat operation in logic format.
MNA	Memory Non–existent Address inverter; MNA is high when QM detects an illegal address during the memory access.
ODD	Odd–numbered opcodes.
QCK	Decoding indicating the QCK state of QC sequencer; when QC reaches this point in its processing, it is almost finished and KM considers this the signal to continue.
RKM	READY KM sequencer; when high, indicates that a repeat operation is not in progress.
SCA	Select bus register CA for transfer of information; this will make possible the storing of the address of the last operand plus 1 in the Accumulator.
Set KM5	KL signals DONE to KM by advancing logic (setting KM5 sends KM to the KMJ state).
Set CD8	Necessary to enable interrupt processing; T0 looks like N0 without 7CD6 and no interrupt processing is allowed if the last command was an N command.
SM	SuM flip–flop; SM is used by the QS sequencer; SM high indicates summing, low, differencing.
WS	Working Sign flip–flop; WS reflects the sign of the operand stored in OA.

FIGURE 13.3–2 The KM Flow Chart for Controlling Repeat Operations

KM

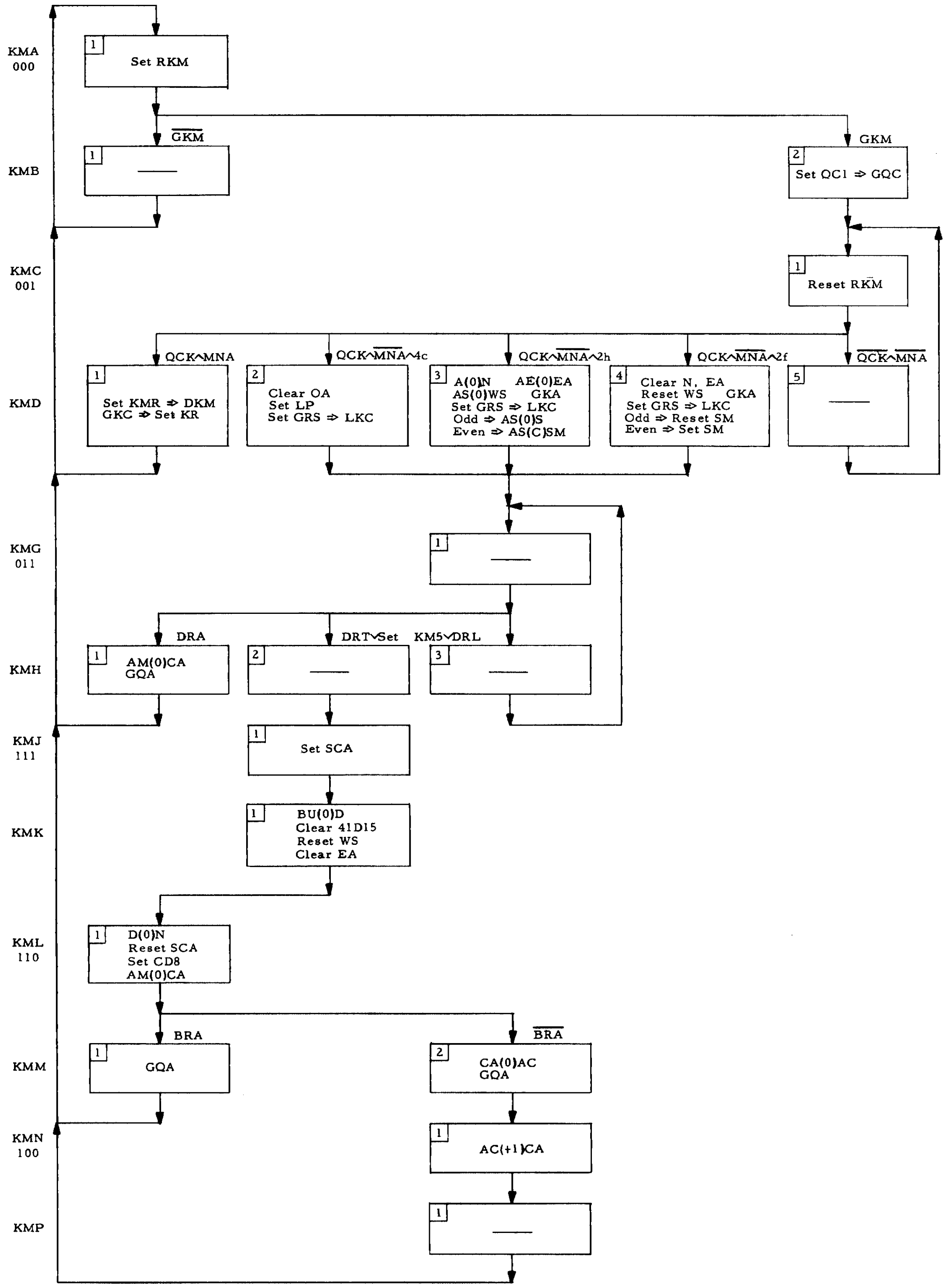
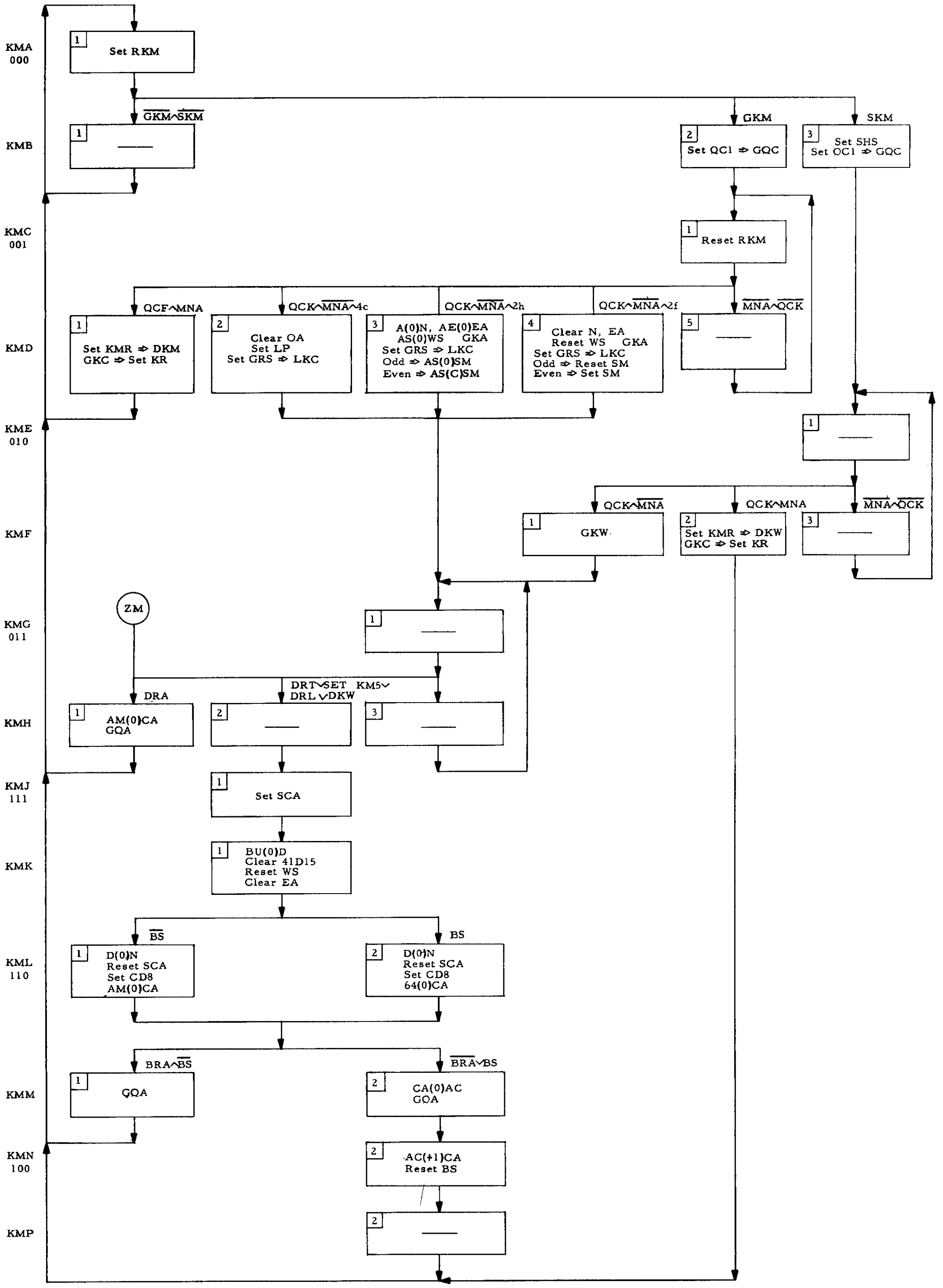


FIGURE 13.3–3 The KM Flow Chart

KM



SECTION 13.4 – THE KJ SEQUENCER

The KJ sequencer, like KT, handles two register and two control opcodes, specifically, R2, R3, X2 and X3. Processing is similar to that carried out by KT, but is considerably more complex.

The control commands X2 and X3 involve modification of the next command address (held in register CA). The X2 command (SKiP) calls for a jump in the program of the number of words specified by the assembled operand. Thus, KJ adds the operand (OA) to the next command address (register CA). Both of these values have zero exponents (the assembled operand is shifted to zero exponent before opcode startup) so that exponent values do not figure in the addition. However, the fact that the skip may be in either direction means that determination of the new address is dependent upon the sign of the operand. Since the QS sequencer is set up to handle sign manipulation, KJ starts QS to form the sum or difference of the two values. At DONE QS , KJ checks the sign of the resulting value for positive before sending it to the CA register as the next command address.

The X3 command (TRansfer and Mark) calls for a transfer in the program to the location specified by the assembled operand with the further provision that the address of the command following the TRM (held in register CA) be saved to enable a return to the point at which the transfer occurred. This address, called the mark, is stored in the word designated by the assembled operand. Program exeuction then continues at the mark plus one. This command is used by programmers to incorporate subroutines in their programs. For example, if the program calls for several determinations of sine values, the programmer will not copy all the lines of code necessary to the performance of this calculation into his program at each iteration; instead, he will include the coding once and, for each use, transfer to the subroutine with the TRM command. At the end of the subroutine, the programmer calls for a return to the mark by means of an X0 command, i.e., GO to the contents of the first word in the subroutine. Since the previous contents of CA are stored in that location, this effects a return to the proper place in the program. Implementation of these actions by KJ is quite straightforward. KC checks the assembled operand for positive sign and shifts it to zero exponent before starting KJ. At GO KJ, the operand is in OA and the address of the next command is in register CA. KJ sends the contents of CA to the B register to be stored in memory and the assembled operand to register CA to be used as the address in the memory WRITE operation. Thus, when QM is started, the former contents of CA are written in the location specified by the assembled operand. QM increments the address in CA so that program execution continues at the mark plus one. Note that the programmer must not use the first location in the subroutine since that word is used to store the mark.

KJ also handles the storing of the program mark during the processing of interrupts. At KCA, the occurrence of an enabled interrupt request causes entry into the interrupt branch of Master Control unless the last command left useful information in OA. (The processing of an interrupt before this information is used would result in less of the information.) Master Control starts KJ for the purpose of storing the program mark and setting up entry into the Interrupt Service Routine. The entry point to this routine does not change; it is always location 64_{10}. KJ supplies the necessary transfer and mark to this location, handling it as a pseudo–TRM command. The distinction between the handling of TRM and interrupts by KJ is simply that, for interrupts, 64 is sent to CA rather than the assembled operand. When the mark has been stored, KC will access the command in location 65 and, thus, start execution of the Interrupt Service Routine.

The two register commands handled by KJ (R2 and R3) both call for extraction of the contents of the indicated register using the assembled operand as the extractor with the result going to OA for R2 and to the Accumulator for R3. Recall that the KT sequencer does not use the Adder in performance of the extraction required by the R1 command. The purpose of this command is to modify the contents of the indicated register so that the result of the extraction is left in the register and use of the Adder is not required. For the R2 and R3 commands, however, the value in the register is undisturbed by the operation; the extractions are being used to read the information in the indicated register. Thus, the Adder is used to perform the extraction, the result is gated into S, then to N, and, in the case of ERA, the QA sequencer is started to store it in the Accumulator.

FIGURE 13.4—1 Algorithm for the Processing of R2 and R3 Commands by KJ

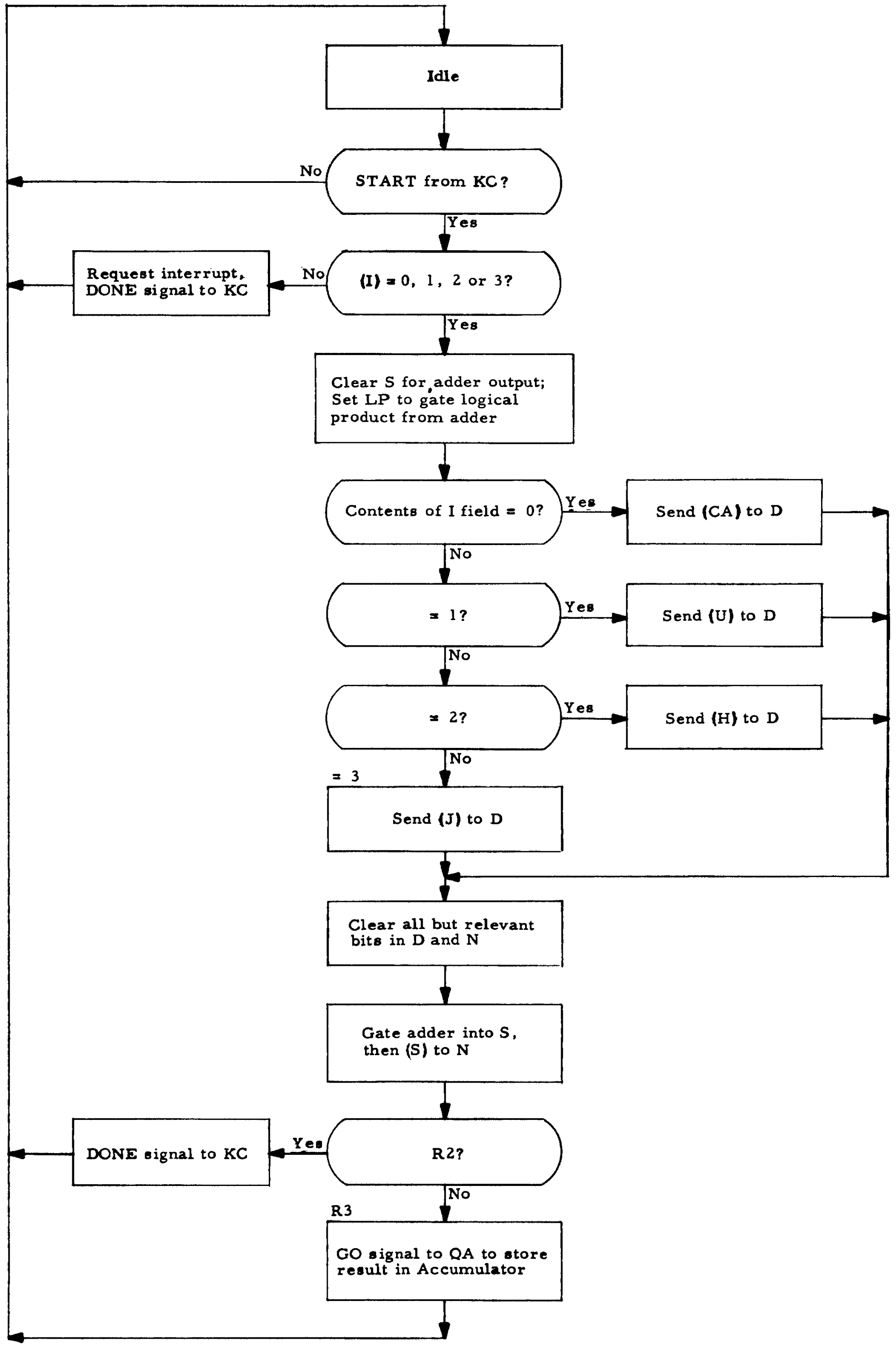

Idle
No
START from KC?
Yes
Request interrupt, DONE signal to KC
No
(I) = 0, 1, 2 or 3?
Yes
Clear S for adder output; Set LP to gate logical product from adder
Contents of I field = 0?
Yes
Send (CA) to D
No
= 1?
Yes
Send (U) to D
No
= 2?
Yes
Send (H) to D
No
= 3
Send (J) to D
Clear all but relevant bits in D and N
Gate adder into S, then (S) to N
DONE signal to KC
Yes
R2?
No
R3
GO signal to QA to store result in Accumulator

TABLE 13. 4–1 Terms Used on the KJ Flow Chart for R2 and R3 Commands

ABR	Addressable Bus Register inverter; ABR is high when decoding of information in the index field indicates that a legal register address has been used.
BU	BU register; after bus register is selected (XZ, SRU, SRH, SRJ) BU is used to indicate that register. Thus, SRJ followed by BU(0)14D0 causes transmission of the contents of register J to register D.
GQA	GO signal to QA sequencer.
KR	Opcode DONE to KC sequencer.
R2, R3	Register commands ERO (Extract Register into OA) and ERA (Extract Register into Accumulator).
Set LP	Set the LP flip–flop to cause the Adder to form the logical product of the operands in registers N and D.
SKJ	START signal to KJ sequencer for commands R2 and R3.
SRH, SRJ, SRU, SCA	Select Register H, J, U or CA respectively.
SZ	S Register Zero signal; SZ is high when the S register contains zeros.
XZ	Index field Zero inverter; XZ high indicates that the index portion of the command word (bits 20–15) contains zeros. This is the designation for selecting register CA. Decoding is actually from 5CD0.

'IGURE 13.4–2 The KJ Flow Chart for the Processing of R2 and R3 Commands

KJ

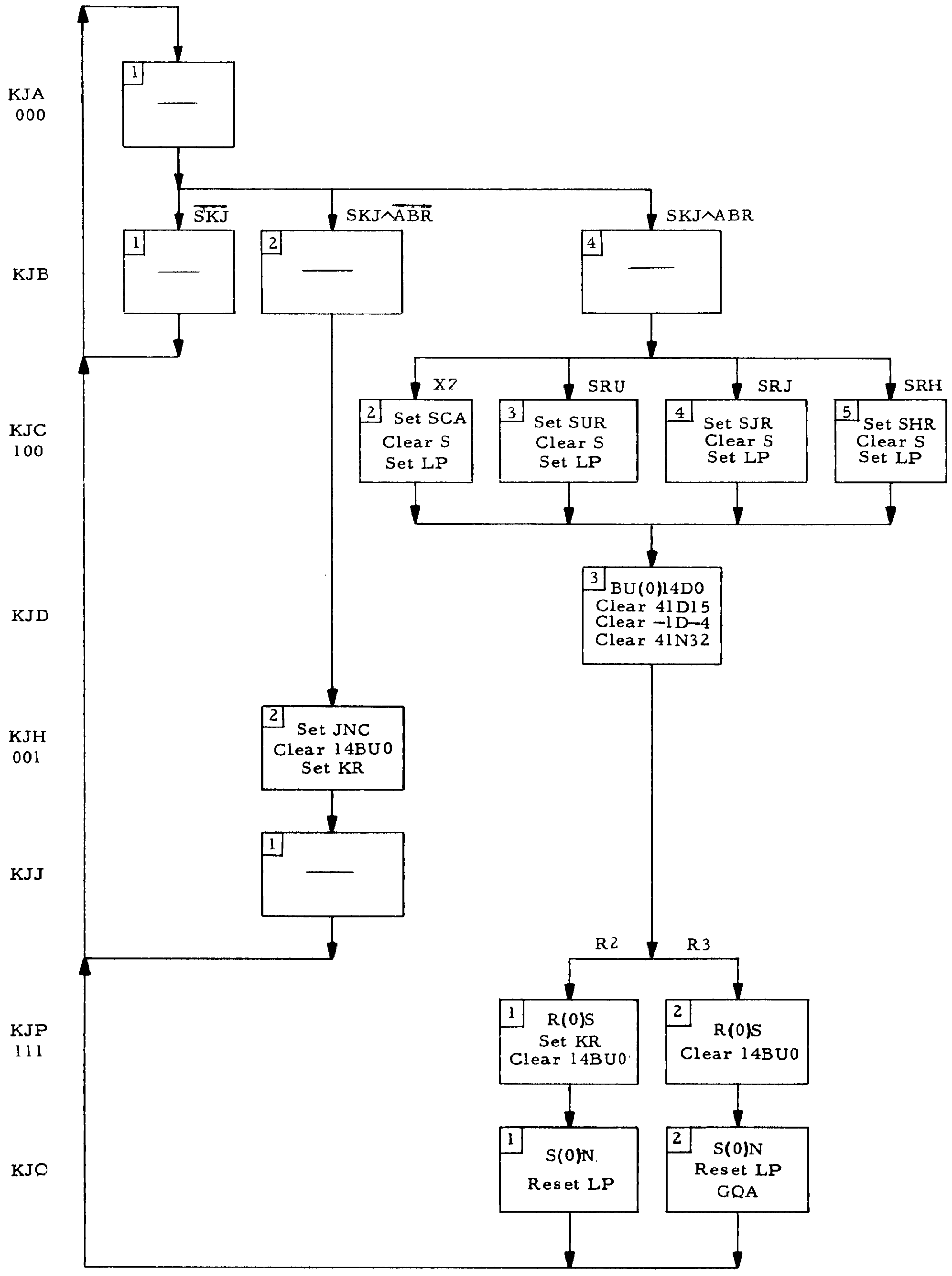


FIGURE 13.4–3 Algorithm for the Processing of X2 Commands by KJ

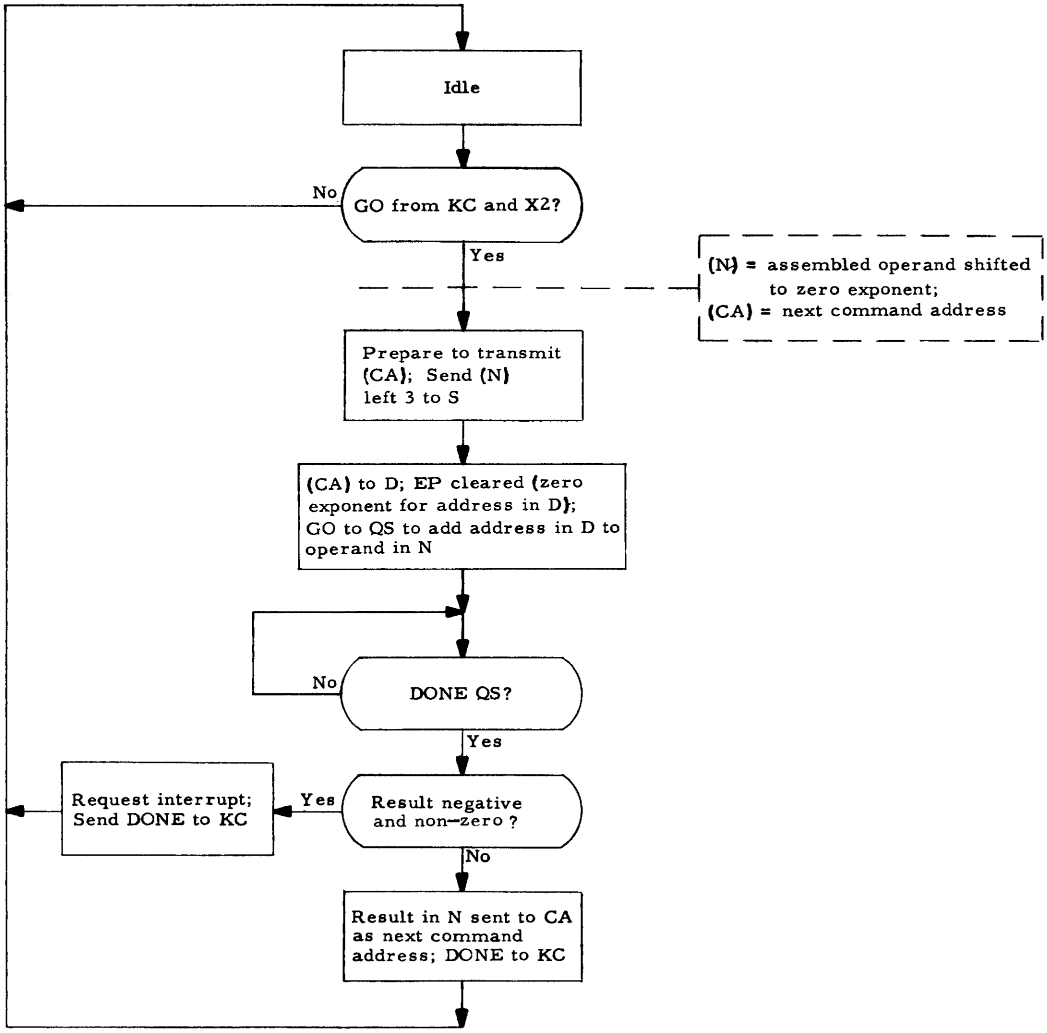

Idle
GO from KC and X2?
No
Yes
(N) = assembled operand shifted to zero exponent;
(CA) = next command address
Prepare to transmit (CA); Send (N) left 3 to S
(CA) to D; EP cleared (zero exponent for address in D); GO to QS to add address in D to operand in N
DONE QS?
No
Yes
Result negative and non–zero ?
Yes
Request interrupt; Send DONE to KC
No
Result in N sent to CA as next command address; DONE to KC

TABLE 13.4–2 Terms Used on the KJ Flow Chart for the X2 Command

BU	BU register; after bus register is selected (XZ, SRU, SRH, SRJ) BU is used to indicate that register. Thus, SRJ followed by BU(0)14D0 causes transmission of the contents of register J to register D.
DQS	DONE QS sequencer.
GKJ	GO signal to KJ sequencer for commands X2 and X3 or for the processing of an enabled interrupt.
GQS	GO signal to QS sequencer.
Int	Indicates that an enabled interrupt is being processed; interrupts are treated as pseudo–X3 commands.
KR	Opcode DONE to KC sequencer.
SRH, SRJ, SRU, SCA	Select Registers H, J, U or CA respectively.
SZ	S Register Zero signal; SZ is high when the S register contains zeros.
WG2	Wait Gate signal; WG2 is brought high as part of the GO QS signal. This inverter is not the echo of WG1.
WS	Working Sign flip–flop; WS holds the sign of the operand stored in OA. WS low indicates a negative sign.
ZE	Zero Exponent signal; in KJ, ZE comes high when the EA register contains zeros.

FIGURE 13.4–4 The KJ Flow Chart for the Processing of X2 Commands by KJ

KJ

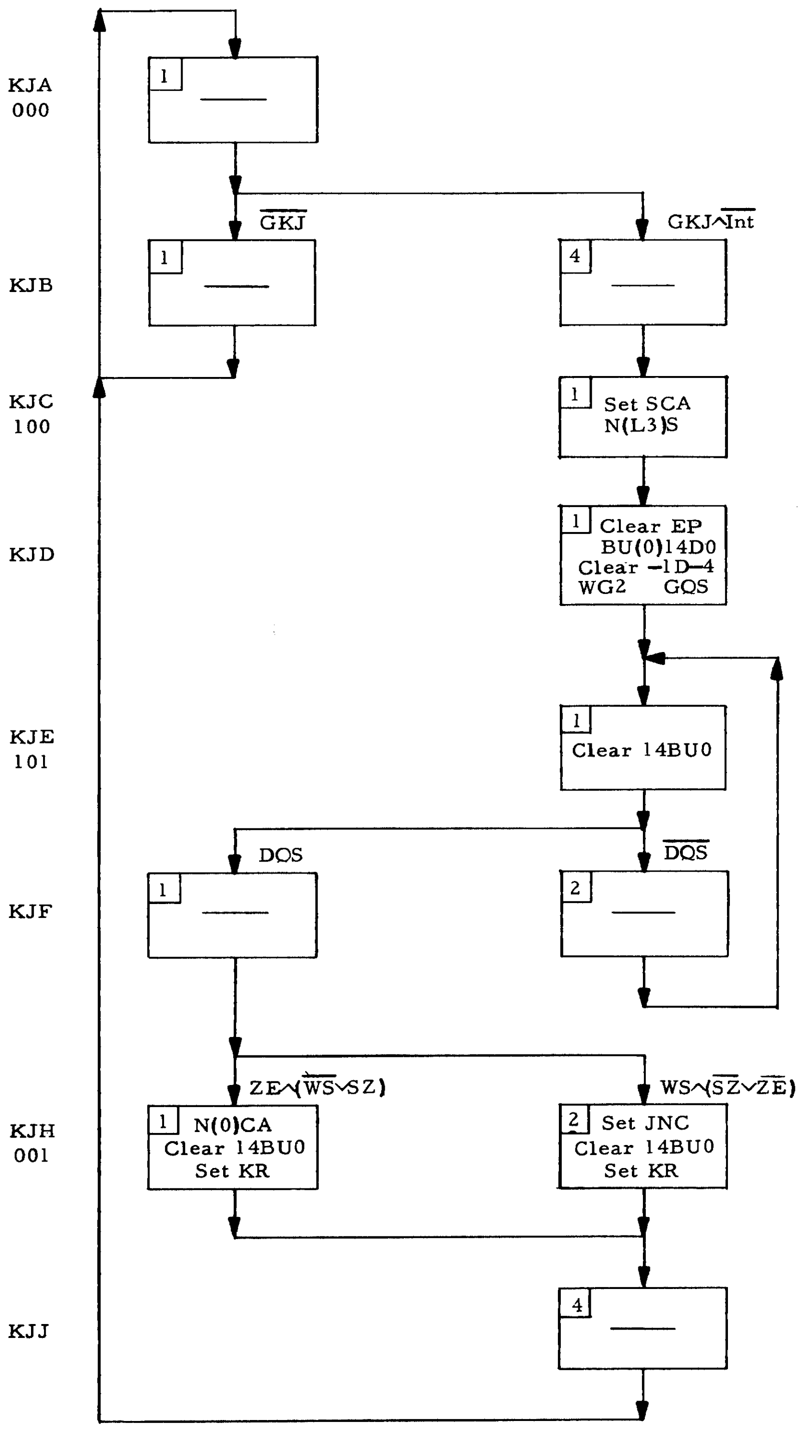


FIGURE 13.4–5 Algorithm for Processing of the X3 Command and Enabled Interrupt Requests by KJ

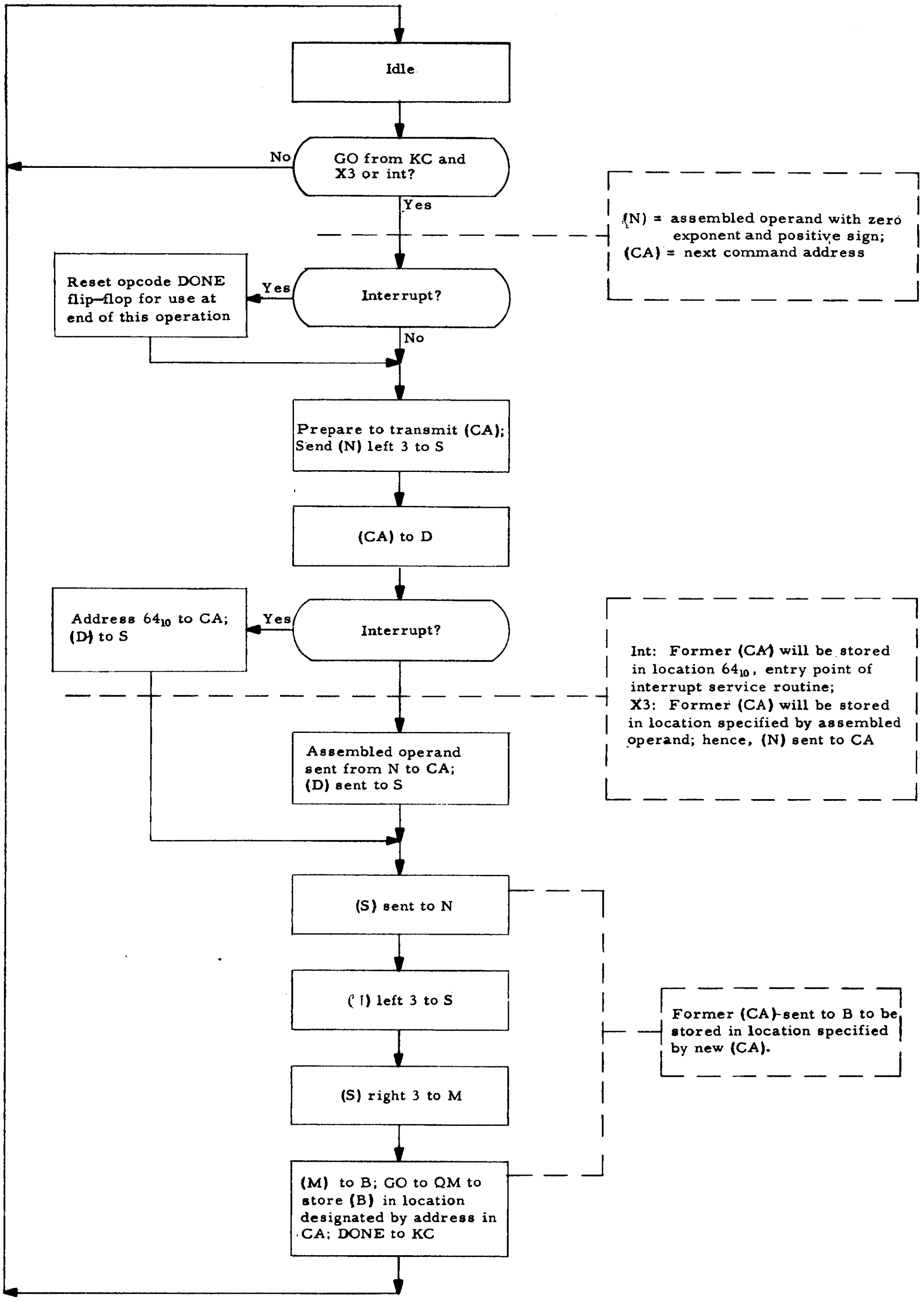

Idle
No
GO from KC and X3 or int?
Yes
(N) = assembled operand with zero exponent and positive sign;
(CA) = next command address
Reset opcode DONE flip–flop for use at end of this operation
Yes
Interrupt?
No
Prepare to transmit (CA); Send (N) left 3 to S
(CA) to D
Address 64_{10} to CA; (D) to S
Yes
Interrupt?
Int: Former (CA) will be stored in location 64_{10}, entry point of interrupt service routine;
X3: Former (CA) will be stored in location specified by assembled operand; hence, (N) sent to CA
Assembled operand sent from N to CA; (D) sent to S
(S) sent to N
(N) left 3 to S
(S) right 3 to M
Former (CA) sent to B to be stored in location specified by new (CA).
(M) to B; GO to QM to store (B) in location designated by address in CA; DONE to KC

TABLE 13.4—3 Terms Used on the KJ Flow Chart for the X3 Command and Interrupts

BU	BU register; after bus register is selected (XZ, SRU, SRH, SRJ) BU is used to indicate that register. Thus, SRJ followed by BU(0)14D0 causes transmission of the contents of register J to register D.
GCA	GO signal to QM sequencer.
GKJ	GO signal to KJ sequencer for commands X2 and X3 or the processing of an enabled interrupt.
Int	Indicates that an enabled interrupt is being processed; interrupts are treated as pseudo—X3 commands.
KCE, KCF	States in the KC sequencer; decoding of these states in conjunction with KJC and KJD indicates that an interrupt is being processed.
KR	Opcode DONE to KC sequencer.
RQM	READY QM sequencer.
SRH, SRJ, SRU, SCA	Select Register H, J, U or CA respectively.
X2, X3	Transfer commands SKP (SKiP) and TRM (TRansfer and Mark).

FIGURE 13.4–6 The KJ Flow Chart for the Processing of X3 Commands and Enabled Interrupt Requests by KJ

K J

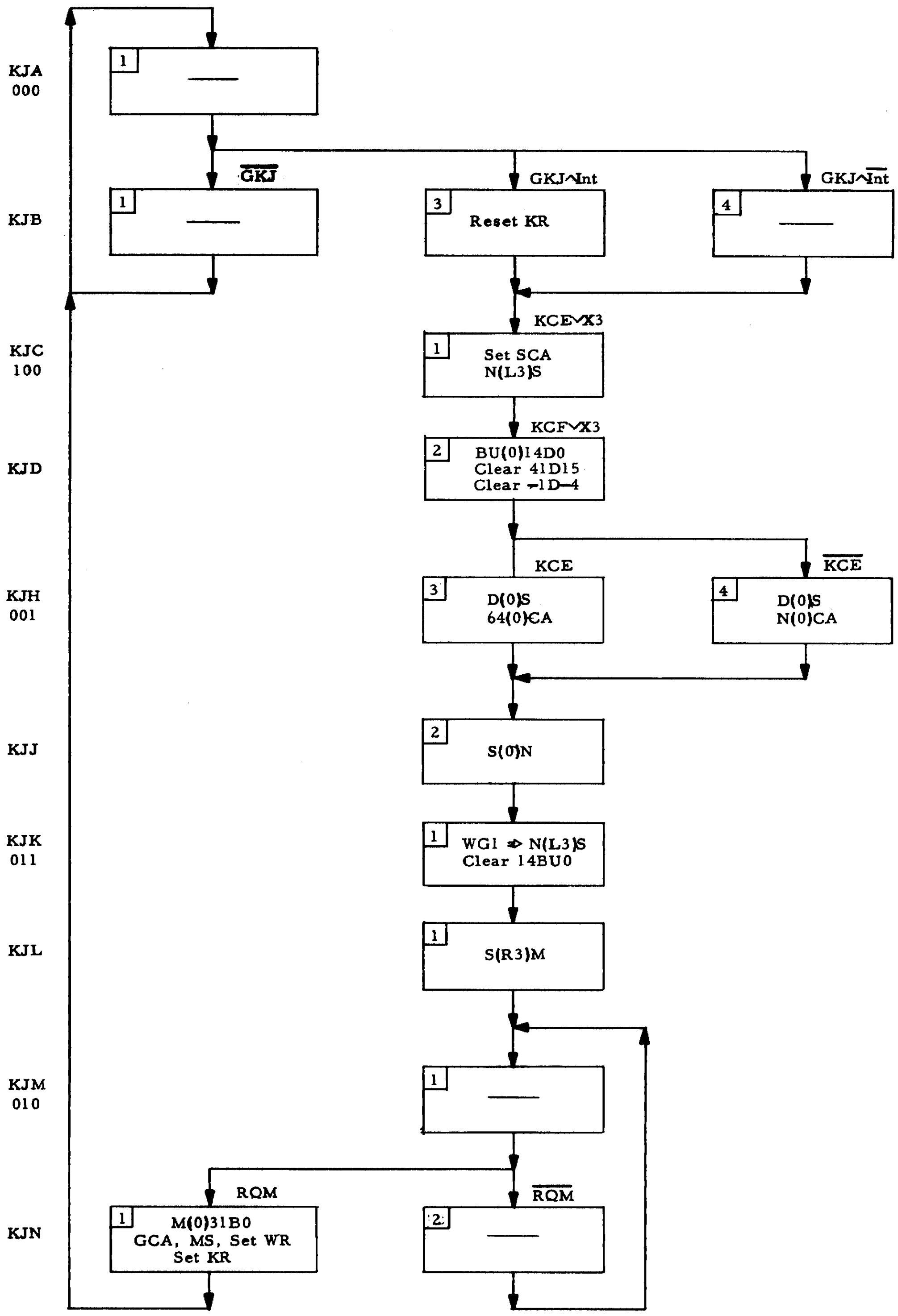
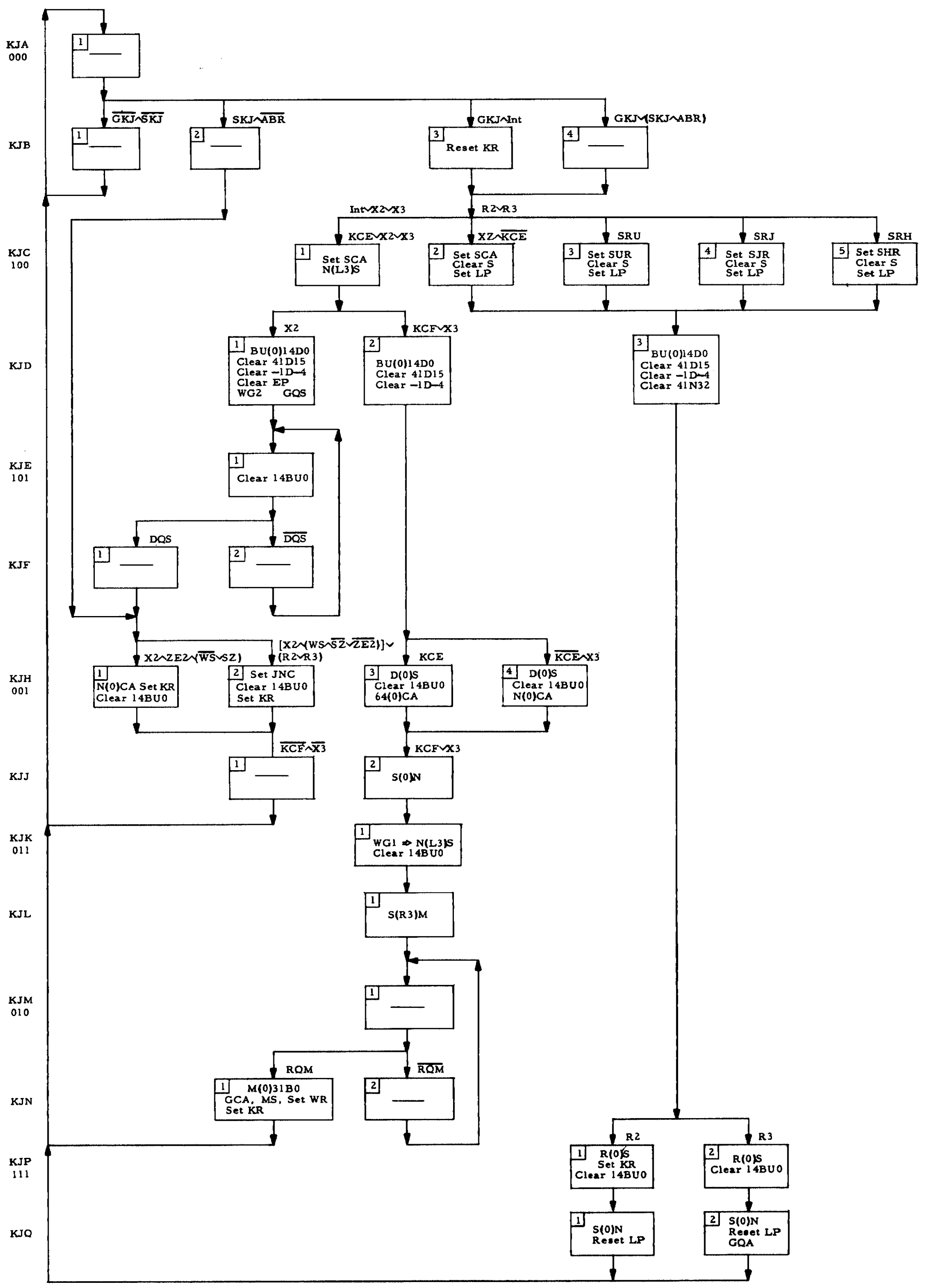


TABLE 13.4—4 Terms Used on the KJ Flow Chart

ABR	Addressable Bus Register inverter; ABR is high when decoding of information in the index field indicates that a legal register address has been used.
BU	BU register; after bus register is selected (XZ, SRU, SRH, SRJ) BU is used to indicate that register. Thus, SRJ followed by BU(0)14D0 causes transmission of the contents of register J to register D.
DQS	DONE QS sequencer.
GCA	GO signal to QM sequencer.
GKJ	GO signal to KJ sequencer for commands X2 and X3 or for the processing of an enabled interrupt.
GQS	GO signal to QS sequencer.
GQA	GO signal to QA sequencer.
Int	Indicates that an enabled interrupt is being processed; interrupts are treated as pseudo—X3 commands.
KCE, KCF	States in the KC sequencer; decoding of these states in conjunction with KJC and KJD indicates that an interrupt is being processed.
KR	Opcode DONE to KC sequencer.
R2, R3	Register commands ERO (Extract Register into OA) and ERA (Extract Register into Accumulator).
RQM	READY QM sequencer.
Set LP	Set the LP flip—flop to cause the Adder to form the logical product of the operands in registers N and D.
SKJ	START signal to KJ sequencer for commands R2 and R3.
SRH, SRJ, SRU, SCA	Select Register H, J, U or CA respectively.
SZ	S Register Zero signal; SZ is high when the S register contains zeros.
WG1	Wait Gate signal; WG1 high is used as a signal to the QS idle state to enable the N(L3)S path.
WG2	Wait Gate inverter; WG2 high is used as part of the GO QS signal.
WS	Working Sign flip—flop; WS holds the sign of the operand stored in OA. WS low indicates a positive sign.
X2, X3	Transfer commands SKP (SKiP) and TRM (TRansfer and Mark).
XZ	Index field Zero flip—flop; XZ is set to indicate that the index portion of the command word (bits 20—15) contains zeros. This is the designation for selecting register CA. Decoding is actually from 5CD0.
ZE	Zero Exponent signal; in KJ, ZE is brought high when the EA register contains zeros.

FIGURE 13.4–7 The KJ Flow Chart

KJ



SECTION 13.5 – THE KL SEQUENCER

The KL sequencer processes all of the L and S commands that call for logical operations. This excludes L2, L3, S2 and S3 which require arithmetic operations and are, therefore, processed by KA.

The commands that call for startup of the KL sequencer are gated through Master Control by two terms: 4c and LP. The 4c term gates those commands when written in mode 0 or 1; the LP flip–flop is set to gate the final operand access for those commands written in mode 2 or 3 and to gate the accessing of the operands during repeat operations. The distinction is necessary due to the fact that all operand assembly occurs in number format. In modes 0 and 1, there are no further accesses. In modes 2 and 3, the assembled operand is used as an address and final access is carried out in logic format for logic commands. Only this last access can be conditioned by the opcode. Thus, if the command is written in mode 0 or 1, the operand is in number format at the time KL is started. Since KL expects to receive an assembled operand that has an exponent value of zero, the exponent is checked by KC before a mode 0 or 1 startup occurs (gated by 4c). The operand is shifted to zero exponent if necessary; truncation to 32 bits is performed by the KL sequencer.

On the other hand, accesses of the final operands for mode 2 or 3 commands, or of the next operand in a block operation, involve no operand assembly and, thus, can be carried out in logic format. These accesses are gated by the signal from the LP flip–flop which is set during the KZC (mode 2 or 3) loop of Master Control or by the KM or KL sequencer during repeat operations. When LP is set, the least significant 32 bits of the operand are sent to register D and exponent register EP is cleared. Thus, no exponent check is necessary and LP gates immediate startup of KL at KCT.

The LP flip–flop serves a different purpose when operand access is not in progress. In the processing of the L4, L5, S4 and S5 opcodes, it directs the Adder to form the Boolean product of (N) and (D). This amounts to the determination of the logical product (extraction) of the two operands. (This process calls for a 1 bit in the result wherever there is a 1 bit in both operands.) Thus, LP is reset at KLD for all but the 4 and 5 opcodes. It is reset at KLK, after the result has been gated from the Adder, for all cases except incomplete repeat operations. On repeat, an LKC signal is sent to KC directing the access of the next operand. The LP signal gates the logic formatting of the accessed information and the startup of KL.

KL is started by KC, both for repeat and non–repeat operations, by means of a GO KL signal. For the two–operand commands (those numbered 4, 5, 6, 7) the Accumulator value is checked for zero exponent. This check is made through use of the subtractor–comparator circuitry which compares the exponent in EP (known to be zero since the assembled operand has either been accessed with a zero exponent or shifted to zero exponent prior to the startup of KL) with that in EA (the Accumulator exponent has been sent from AE to EA). The signal EE is high when (EA) = (EP) so that, in this case, EE signals zero exponent. If EE is not high, the QZ sequencer is started to shift the operand to zero exponent. Following this, both operands are truncated to 32 bits.

Because KL handles all opcodes similarly, it serves little purpose to flow chart the basic algorithm. It is more important to compare the requirements of the opcodes and to understand the subtleties in the logic that adapt it to each case. Referring to Table 13.2–1, the following generalizations concerning the processing of logic opcodes can be made:

1) The operands are handled in the logic format: zero exponent,

truncation to 32 bits. For those operands gated by 4c, a check is made for zero exponent before KL is started; the operands are truncated to 32 bits by KL. For those gated by LP, the operand is accessed in logic format (a zero exponent is sent to register EP and the least significant 32 bits of the operand are sent to register D). For the L4–7 and S4–7 commands, the operand stored in the Accumulator is checked for zero exponent; if it is not zero, QZ is started to shift it to that condition.

2) The odd–numbered commands operate on ((Acc $\overline{\$\ X}$)where $\overline{X}$ is the complement of X and $ indicates a logical operation. (The complement of a value is the logical analog of negation.)

3) The test conditions are satisfied (take next command) if the result is equal to zero.

In all cases, KL gates the result of the operation from the Adder by means of the R(L1)S path. The requirements of each logical operation and the Adder initialization are shown in Table 13.5–1.

In group I, the operand is summed with a zero value and is, thus, unchanged. The group II opcodes call for extraction (the formation of the Boolean product). This is obtained when the LP flip–flop is set. For the opcodes in group III union results from storing one of the operands in S, then transferring the other into S from the Adder. (1 bits already in S will not be destroyed by this single–sided transfer, while additional 1 bits from the other operand will be copied, thus effecting the union.)

The L commands call for storage of the result in the Accumulator. For both repeat and non–repeat operations, the sign left in the AS flip–flop will be that of the operand that was originally stored in the

TABLE 13.5–1 Use of the Adder by KL

Commands:	Single Operand I: L0, L1, S0, S1	Two Operands II: L4, L5, S4, S5	 III: L6, L7, S6, S7
Logic Operation:	Bring	Extraction (1 bit in result in bit positions where <u>each</u> operand has a 1)	Union (1 bit in result in bit positions where <u>either</u> operand has a 1)
Register N:	Cleared	Operand from Accumulator	Cleared
Register D:	Assembled operand (L0, S0) or complement (L1, S1)	Assembled operand (L4, S4) or complement (L5, S5)	Assembled operand (L6, S6) or complement (L7, S7)
Register S:	Cleared	Cleared	Operand from Accumulator
State of LP Flip–Flop:	Reset	Set	Reset

Accumulator for the two–operand commands. For the single–operand commands, the former state of AS is meaningless. Thus, for mode 0 and 1 commands (access in number format) the sign is that of the operand, while for mode 2 and 3 commands (access in logic format) the sign is positive.

On non–repeat tests the contents of the Accumulator are not disturbed when the command is processed. The program continues with execution of the next command or the next command plus 1 depending upon whether or not the result of the operation was equal to zero. For repeat test commands, the address plus 1 of the last operand processed is stored in the Accumulator so that the programmer can determine the location of the operand last processed.

Repeat operations have been described in some detail in Section 13.2. The following points should be emphasized:

1) Following the initial operand access which is set up by KM, all subsequent accesses are controlled by KL. KL clears the N register in anticipation of the startup of QS, sends an LKC signal to KC to direct the access of the next operand, sets the LP flip–flop to gate the access in logic format and resets WS as part of the logic formatting of the result of the previous operation.

2) A Done Repeat Arithmetic operation signal is not generated as shown, but $KLK \wedge \overline{RKM} \wedge CD8 \wedge \overline{RTF}$ has the same affect as does DRA and is replaced on the flow chart by DRA for convenience. (DRA ⇒ AM (0) CA, reset KM1, reset KM3).

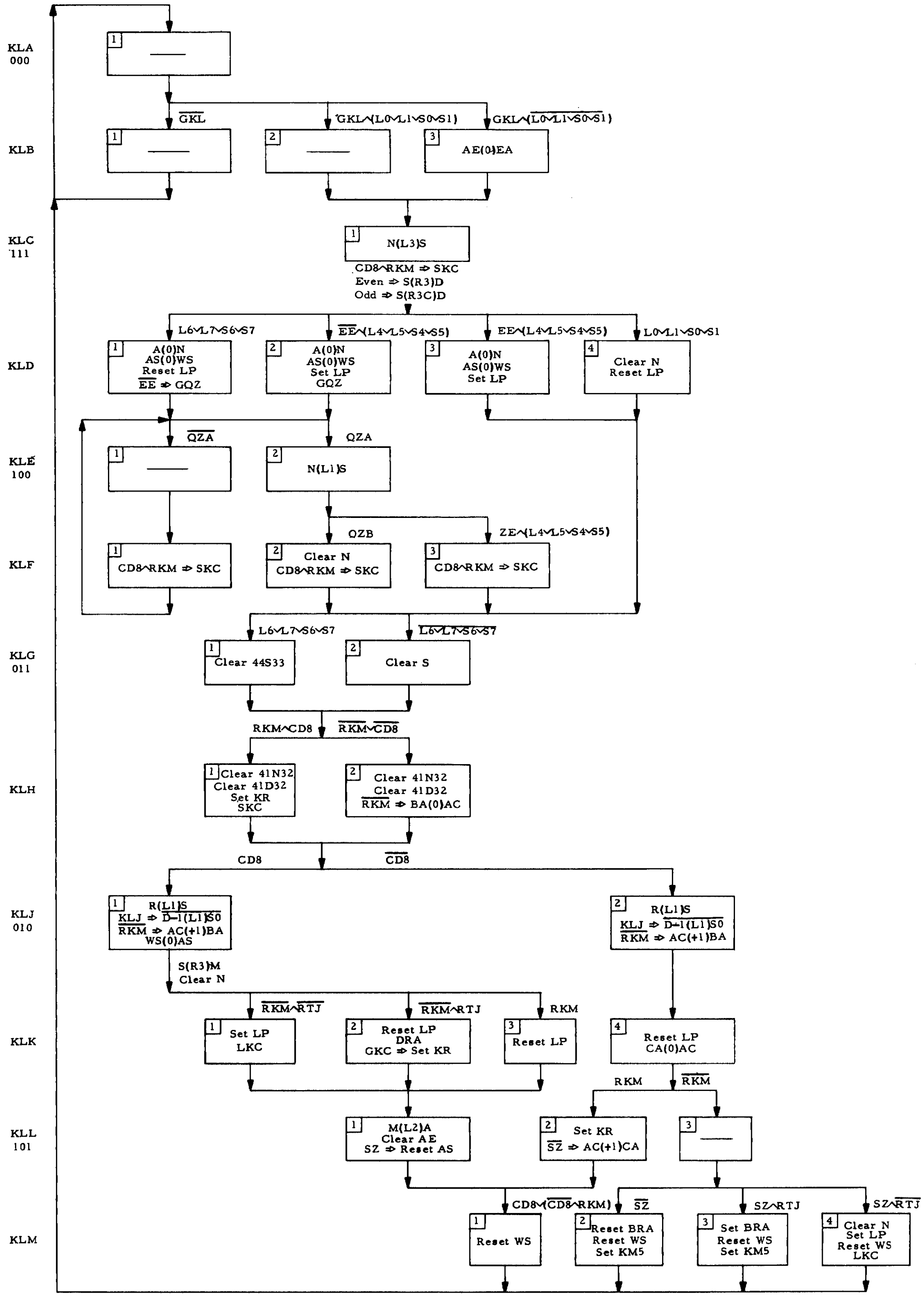
3) KL does not start QA to store the result of each L operation in the Accumulator, but performs these shifts on its own, clearing the exponent register and resetting WS in line with logic format.

TABLE 13.5–2 Terms Used on the KL Flow Chart

AS	Accumulator Sign flip–flop; AS holds the sign of the operand currently stored in the A register.
BRA	BRAnch flip–flop; in repeat operations, if the tests are all satisfied and the operation terminates due to end of block, the BRA flip–flop is set; computation continues with the next command which normally calls for a transfer or branch in the program.
DRA	Done Repeat Arithmetic operation signal; DRA is not generated by KL; rather, the combination of KLK $\wedge$ CD8 $\wedge$ $\overline{\text{RKM}}$ $\wedge$ $\overline{\text{RTF}}$ is equivalent to the DRA signal since both have the following consequence: AM(0)CA, reset KM1, reset KM3.
EE	Exponents Equal signal; EE high indicates that (EA) = (EP).
EVEN	Even–numbered opcodes, i.e., L0, 4, 6, S0, 4, 6.
GKC	GO KC sequencer; GKC advances the KC logic by setting the KR flip–flop.
GKL	GO KL sequencer.
GQZ	GO QZ sequencer.
KLJ	Decoding of KLJ state of the KL sequencer; KLJ decoding inhibits the transfer of the bit held in bit position D–1 of register D to the S register when the Adder output is gated to S. (The path D–1(L1)S0 is inhibited.)
KR	Opcode DONE signal to KC.
L0, L1, L4, L5, L6, L7	Logic commands processed by the KL sequencer: CAL, CCL, EXL, ECL, UNL, UCL.
LKC	Loop KC sequencer; the LKC signal is sent to direct KC to access the next operand in a repeat operation.
LP	Logical Product flip–flop; LP is set during operand access if logic format is required; this happens for the L and S commands processed by KL when written in the repeat mode or in the non–repeat mode with mode 2 or 3 commands. LP is set during the operation of KL for the opcodes numbered 4 and 5 which call for formation of the logical product by the Adder circuitry. For the other opcodes processed by KL, the LP flip–flop is reset.
ODD	Odd–numbered opcodes, i.e., L1, 5, 7, S1, 5, 7.
QZA, QZB	Normal idle states of the QZ sequencer; when QZA and QZB are high, QZ is not in operation.
RKM	READY KM sequencer flip–flop; RKM is set when KM is not in use, reset if KM is in use. Hence, RKM ⇒ no repeat operation, $\overline{\text{RKM}}$ ⇒ repeat.
R(L1)S	Path enabled to gate the Adder output to register S during operation of KL.
RTF	The RTF inverter indicates the opposite state to that of the RTJ inverter.
RTJ	Repeat operation Terminates inverter; during operation of KL, RTJ is high due to the end of the block or the occurrence of an enabled logic flag.
S0, S1, S4, S5, S6, S7	Logic test commands processed by the KL sequencer (IOZ, ICZ, IEZ, ICZ, IUZ, IUC).
Set KM5	KL signals DONE to KM for repeat operations by advancing logic (setting KM5 sends KM to the KMJ state).
SKC	Early Start signal sent to KC; SKC is sent only during non–repeat operations of KL.
SZ	S register Zero signal; SZ is high when the S register contains zeros.
WS	Working Sign flip–flop; WS holds the sign of the operand currently stored in OA.
ZE	Zero Exponent signal; during the operation of KL, ZE is high when the EA exponent register contains a zero exponent.

FIGURE 13.5–1 The KL Flow Chart

KL



SECTION 13.6 – THE QS SEQUENCER

The use of this sequencer was described in detail in Section 13.1. It will suffice to review the points made at that time:

1) QS is started by KC, KA, and KJ to perform floating–point summing and differencing operations on two operands;

2) the user sequencers set up the operands in the N and D registers with exponents in the EA and EP registers respectively. The sign of the operand in register N is the WS flip–flop;

3) at GO QS, the state of the SM flip–flop indicates whether a sum or difference operation is to be performed. (SM high indicates summing operation.) This state is arrived at by means of combining the sign in WS, the sign of the operand held in register D (taken from the B28 flip–flop when the operand is accessed), and the sign of the operation itself (add or subtract);

4) QS handles equalization of exponents if necessary and determination of the final sign value.

At DONE QS the resulting mantissa is stored, rounded if necessary, in register N, the sign is in the WS and SM flip–flops, and the signed exponent is in the exponent circuits as follows. If, at GO QS, the operand in N is equal to zero, an early exit from QS leaves the exponent of the result in registers EP and ES. If this is a non–zero operand, the normal exit from QS leaves the final exponent in register EA and the exponent plus 1 in registers EP and ES. The way in which these values are handled by the user sequencers is shown in Table 13.6–1.

The idle states of the QS sequencer are unusual in two respects: there are three distinct idle loops and a GQS is not sufficient to set the

TABLE 13.6–1 Use of Exponents Generated by QS

QS User Sequencers:	KJ	KA	KC
Exponent Requirements:	Starts QS for SKP command. No exponents are involved since the next command address has no exponent and the assembled operand is shifted to zero exponent prior to opcode startup.	Expects to find assembled operand in OA, N, EA WS.	At opcode startup, some opcodes require exponent of assembled operand in EA, some require exponent of Accumulator operand in EA.
If (N) = 0: (Exponent in EP and ES)		The path ES(0)EA is enabled at KAH if DQS $\wedge$ QSF (DQS is generated at QSF only if (N) = 0.	If DQS available at KCT [implying (N) = 0] requirements of opcode determine which of these paths is enabled: ES(0)EA or AE(0)EA.
If (N) ≠ 0: (Exponent in EA, exponent plus 1 in EP and ES)			The path ES(–1)EP is enabled to correct exponent in EP in all cases except those where zero exponent is required.

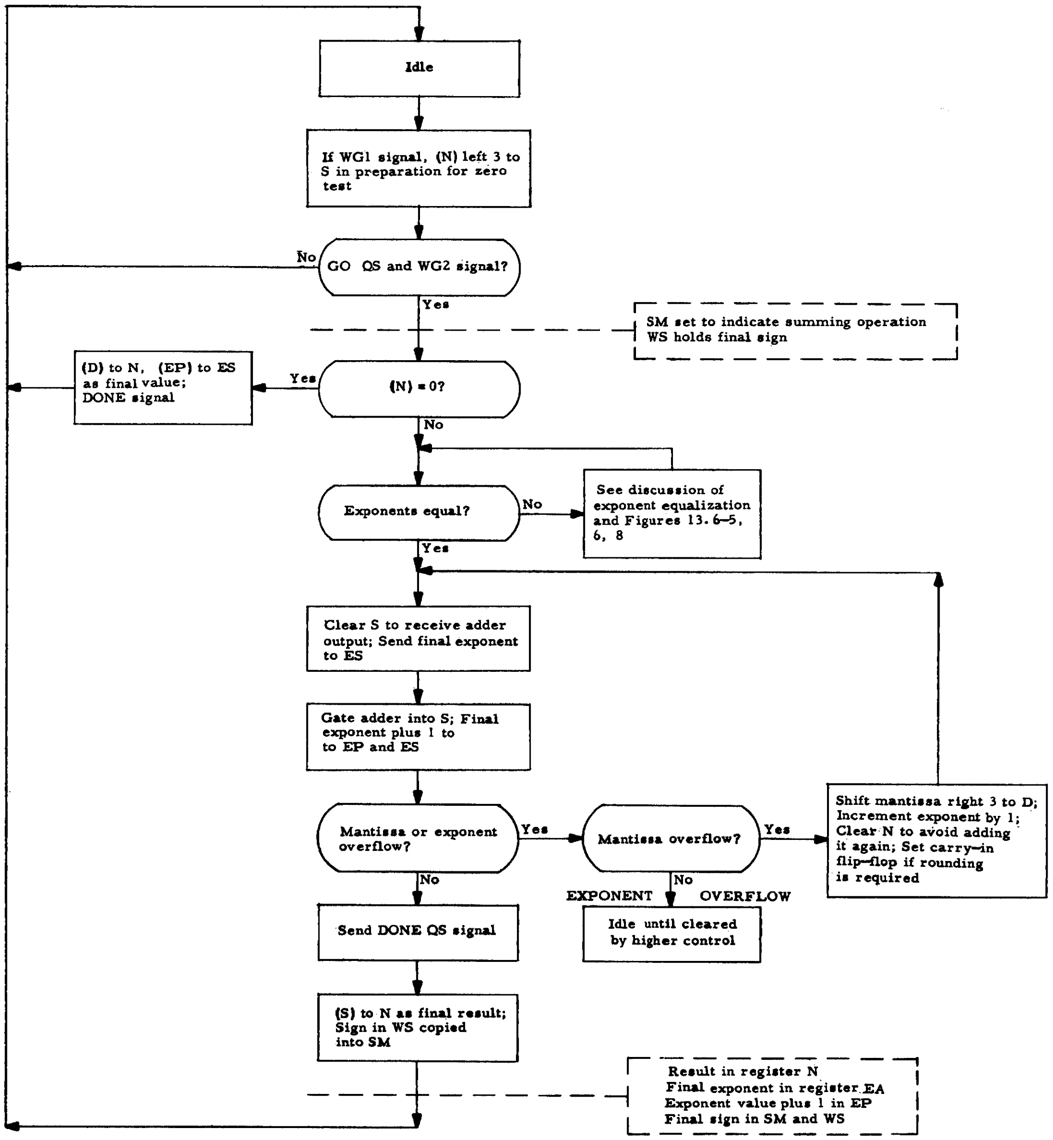
sequencer in operation. Two signals are required for the startup: WG2 and GQS. Any time both of these signals are available, QS exits from idle.

Before exiting from the idle loop, the path N(L3)S is enabled to send the operand to register S for a zero test on (N). This enabling can be handled by means of a WG1 signal sent to QS or, as in the case of KJ, by direct enabling of the path. Note that WG2 does not echo WG1. This makes it possible to use the WG1 signal for enabling of the N(L3)S path even when the QS sequencer is not about to be started, a circumstance which is necessary to circumvent more direct use of this overloaded enable path.

Figure 13.6–1 shows the algorithm for a summing operation (SM high) when (N) = 0 or the exponents of the two operands are equal; Figure 13.6–2 contains the corresponding portion of the QS flow chart. A mantissa overflow, indicated by signal S42, is corrected by means of a right–3–bit shift with a corresponding increment of 1 to the exponent. Register N is cleared so that the operand stored there won't be added again and the sum is sent to D for gating through the Adder again. If the round–up inverter is high enabling of the RU(0)CZ path sets the carry–zero flip–flop to cause rounding this time through the Adder. Due to the nature of binary arithmetic, the mantissa will never overflow by more than 1 bit. Thus, this shift will always correct the condition. Exponent overflow (generation of an exponent larger than 77_8) is not handled by QS. Rather, QS idles until cleared by KC or KA, whichever started the operation. The control sequencer will request the interrupt and take care of any other book-keeping operations necessary to its own return to idle.

When the SM flip–flop is reset, indicating a differencing operation, the operand in register N is 1's complemented and an add cycle performed. The usual case is taken to be that where /D/>/N/ so that complementing

FIGURE 13.6–1 Summing Algorithm


Idle
If WG1 signal, (N) left 3 to S in preparation for zero test
No
GO QS and WG2 signal?
Yes
SM set to indicate summing operation
WS holds final sign
(D) to N, (EP) to ES as final value; DONE signal
Yes
(N) = 0?
No
Exponents equal?
No
See discussion of exponent equalization and Figures 13.6–5, 6, 8
Yes
Clear S to receive adder output; Send final exponent to ES
Gate adder into S; Final exponent plus 1 to to EP and ES
Mantissa or exponent overflow?
Yes
Mantissa overflow?
Yes
Shift mantissa right 3 to D; Increment exponent by 1; Clear N to avoid adding it again; Set carry–in flip–flop if rounding is required
No
No
EXPONENT OVERFLOW
Send DONE QS signal
Idle until cleared by higher control
(S) to N as final result; Sign in WS copied into SM
Result in register N
Final exponent in register EA
Exponent value plus 1 in EP
Final sign in SM and WS

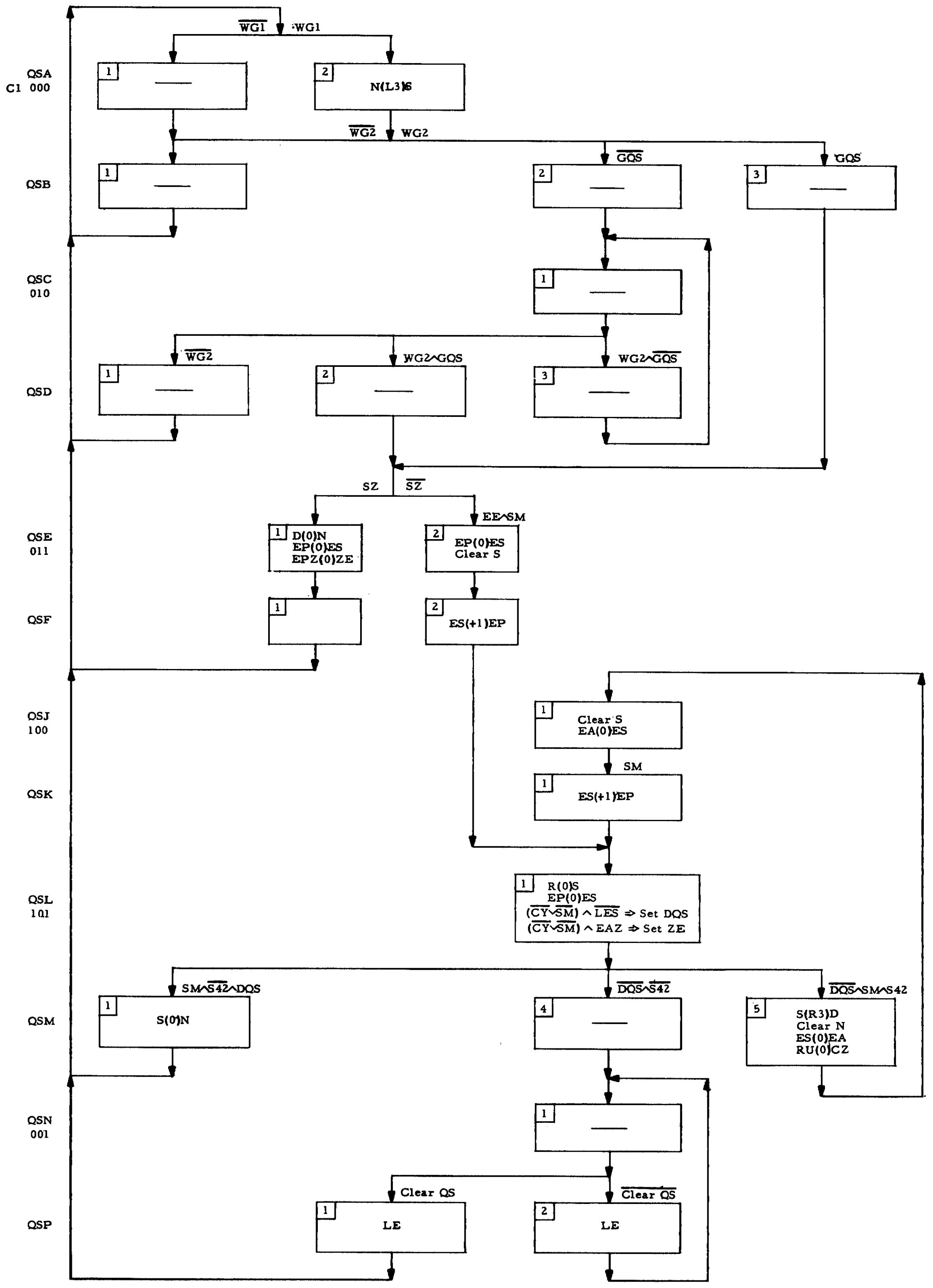
(N) and adding (N) to (D) results in generation of a carry out. In binary arithmetic, this carry out indicates that the result of the operation is in true form (since the uncomplemented value is larger) but is low by 1. Thus, a carry in to the zero order of the Adder is necessary to complete the operation. Since the usual case is taken to be that in which a carry in is necessary, the carry in flip–flop (CZ) is set before the result of any differencing operation is gated from the Adder.

TABLE 13.6–2 Terms Used on the QS Flow Chart for Summing

Clear QS	Signal from higher level control. On detection of exponent overflow by QS, user sequencer requests an interrupt, clears QS, and takes any actions necessary to its own status. QS, on being sent a Clear signal, returns to idle.
CY	CarrY inverter; CY high indicates that a mantissa overflow has occurred in the Adder output. CY, if enabled (CYE high), sets S42 or S43 depending on the output path from the Adder.
CYE	CY Enable flip–flop.
CZ	Carry Zero flip–flop; CZ high indicates a carry into the zero order of the Adder. It is used in summing to round up the mantissa if RU is set.
DQS	DONE QS sequencer.
EAZ	EA register Zero inverter; when high, EAZ indicates that the exponent held in register EA is zero.
EE	Exponents Equal signal; EE indicates (EA) = (EP).
EPZ	EP register Zero inverter; when high, EPZ indicates that the exponent held in register EP is zero.
GQS	GO signal to QS sequencer; GQS must occur when WG2 is high in order to start the sequencer.
LE	Large Exponent; LE indicates that an exponent overflow has occurred.
LES	Large Exponent Signal; LES is high when (EE) = 101_8. (This is 77_8 plus 1 since the exponent in EP is one greater than the final exponent when the result is gated from the Adder.)
R(0)S	Path enabled to gate output from Adder.
RU	Round–Up inverter; RU is high during right shifts in accordance with the round–up rule: $RU \Rightarrow S2 \wedge (S0 \vee S1 \vee S3 \vee DB)$.
SM	SuM flip–flop; SM high indicates that a summing operation is called for.
S42	Bit 42 in register S; if the output from the Adder is gated R(0)S, and $CY \wedge CYE$ are high, S42 goes high.
SZ	S register Zero signal; SZ is high when register S holds zeros.
WG2	Wait Gate inverter; WG2 is used in conjunction with GQS to start the QS sequencer.
WS	Working Sign flip–flop; WS holds the sign of the operand in N at GO QS and that of the result at DONE QS.
ZE	Zero Exponent signal; during operation of QS, ZE comes high when the final exponent is zero.

FIGURE 13.6–2 The QS Flow Chart for Summing

QS



Consider the following example. To store the quantity –22 in the Accumulator, the A1 command (CLear Subtract) is written with an operand of 22. This sends 22 to register A, zero to the AE register, and the AS flip–flop is set to indicate negative. If this is followed by an A3 command (SUB) with an operand of –31, then operand assembly will store 31 in register N, zero in register EA, and the SM and WS flip–flops will hold the negative sign. The KA sequencer then sends the operand from the Accumulator to OA, and that from OA to registers D and EP. The state of SM is fixed according to the rule demonstrated in Table 13.1–2. (A subtract operation with negative values in AS and SM leaves SM low to indicate a differencing operation. WS remains high; its final value will be that of the larger operand.) Thus, at GQS:

```
             41                       0
register D  |0 . . . . . . . . . 011001|
             41                       0
register N  |0 . . . . . . . . . 010010|
```

SM flip–flop reset; WS flip–flop set.

The operand in register N is complemented and transferred to register D while that from D is sent to N.

```
             41                       0
register D  |1 . . . . . . . . . 101101|
```

As was stated earlier, CZ is set on the assumption that the uncomplemented value (original (D)) has the larger absolute value so that the result will be in normal form, low by 1. With CZ set, the add cycle is as follows:

```
(N) = 000000000000000000000000000000000000011001
(D) = 111111111111111111111111111111111111101101
CZ                                             1
------------------------------------------------
CY    000000000000000000000000000000000000000111 = 7₈.
```

The CY (CarrY out) signal high indicates that the setting of the carry in was correct and that the larger absolute value was, in fact, the original (D). This means that the sign of the value originally stored in register N is not the sign of the result and the state of the WS flip–flop is reversed. Before the operation is complete, the sign is copied into the SM flip–flop. Thus, at DQS, WS is low and SM high to indicate that the final value is +7.

For the case where /D/</N/, the output from the Adder is handled differently. If the values used in the example above were reversed, the operation would be handled as follows:

```
                41                    0
   register D  [0 . . . . . . . . . 10010]
                41                    0
   register N  [0 . . . . . . . . . 011001]
```

Contents of register N are complemented and transferred to register D while (D) are sent to N.

```
                41                    0
   register D  [1 . . . . . . . . . 100110]
```

The addition that takes place is as follows:

```
(N) = 000000000000000000000000000000000000010010
(D) = 111111111111111111111111111111111111100110
CZ                                             1
______________________________________________________
 __
 CY   111111111111111111111111111111111111111001
```

No carry out is generated. This indicates that the absolute value of the complemented operand (the original (D)) is larger. Therefore, the result is in 2's complement form and the carry in flip–flop should not be set. QS checks the state of the CY flip–flop before gating the result from the Adder. If CY is low, QS resets the carry in flip–flop (CZ) and re–enters the Adder portion of its logic. The result gated from the Adder will be the 1's complement of the correct result.

Thus:

(S) = 111111111111111111111111111111111111111111000

This value is 1's complement to give the final answer: 7_8. In this case, the larger value was originally stored in register N and, thus, the correct final sign is already in the WS flip–flop. This sign is copied into SM so that, at DONE QS, WS is high and SM is low indicating a result of –7. Figures 13.6–3 and 13.6–4 show the algorithm and flow charts for differencing operations.

Before two numbers can be correctly summed or differenced, their exponents must be equal. The EE signal (Exponents Equal) is high when the exponent in the EA register equals that in EP. Since, at GO QS, the operands are stored in registers N and D with exponents in EA and EP respectively, QS uses the EE signal to determine whether or not exponent equalization is necessary. Equalization is carried out by means of a series of mantissa shifts in conjunction with the appropriate exponent increments or decrements. These manipulations are similar to those described for the QZ sequencer with the positioning of mantissas for shifting being determined by the available shift paths. Thus, the mantissa in register N can be shifted left 3 bits at a time while that in register D can be shifted right 3 bits at a time. However, in QZ one of the exponent values is fixed ahead of time and the other is brought into agreement with it whereas, in QS, the equalization algorithm allows for convergence of the two exponent values in order to make possible the retention of maximum possible significance in the mantissas.

The QS equalization algorithm begins with left shifts of (N) along with decrements of the exponent associated with that mantissa. (The mantissa with the larger exponent is held in register N. If this is not the case when exponent equlaization begins, the mantissa in register N

FIGURE 13.6–3 Differencing Algorithm

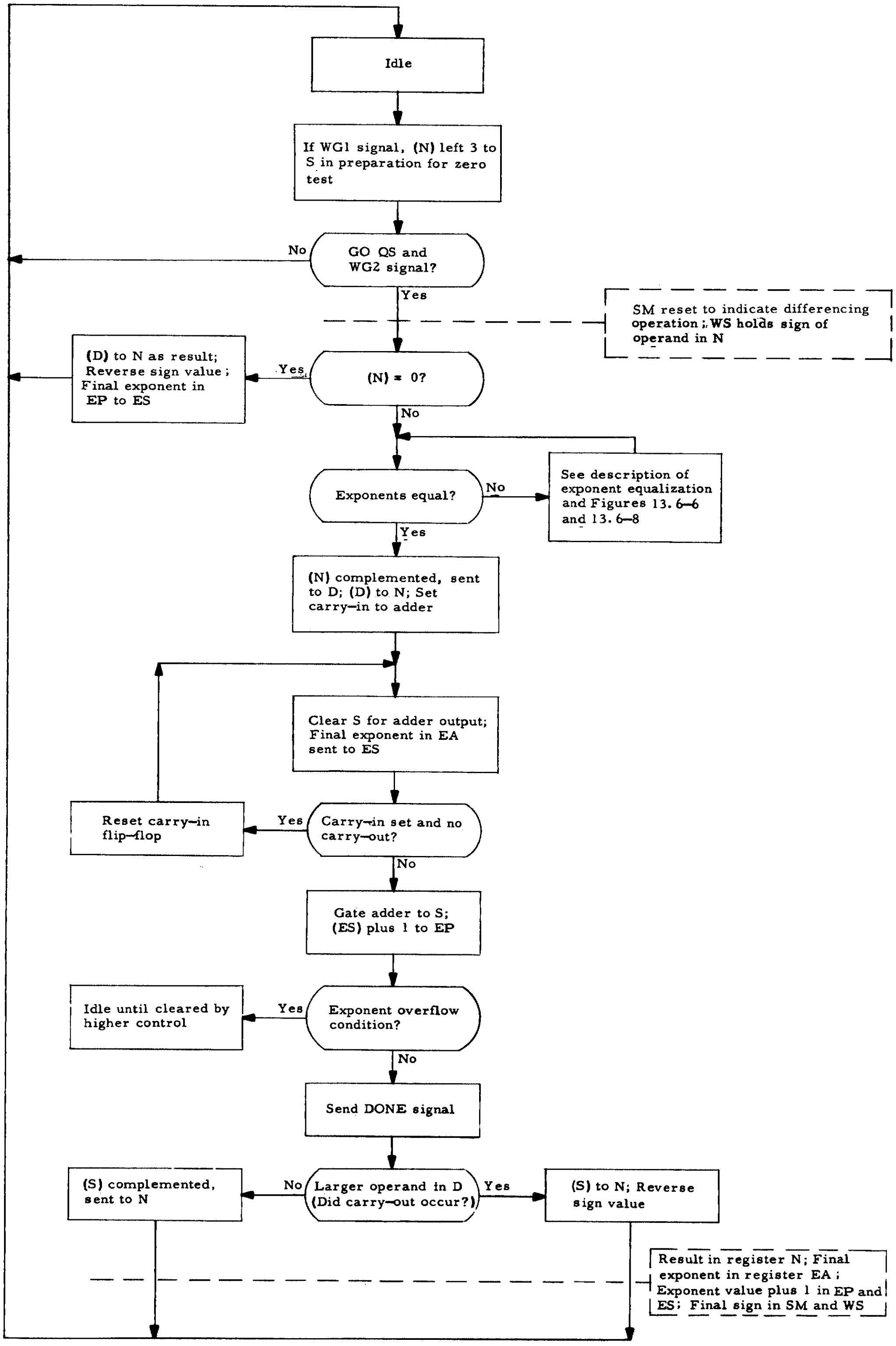
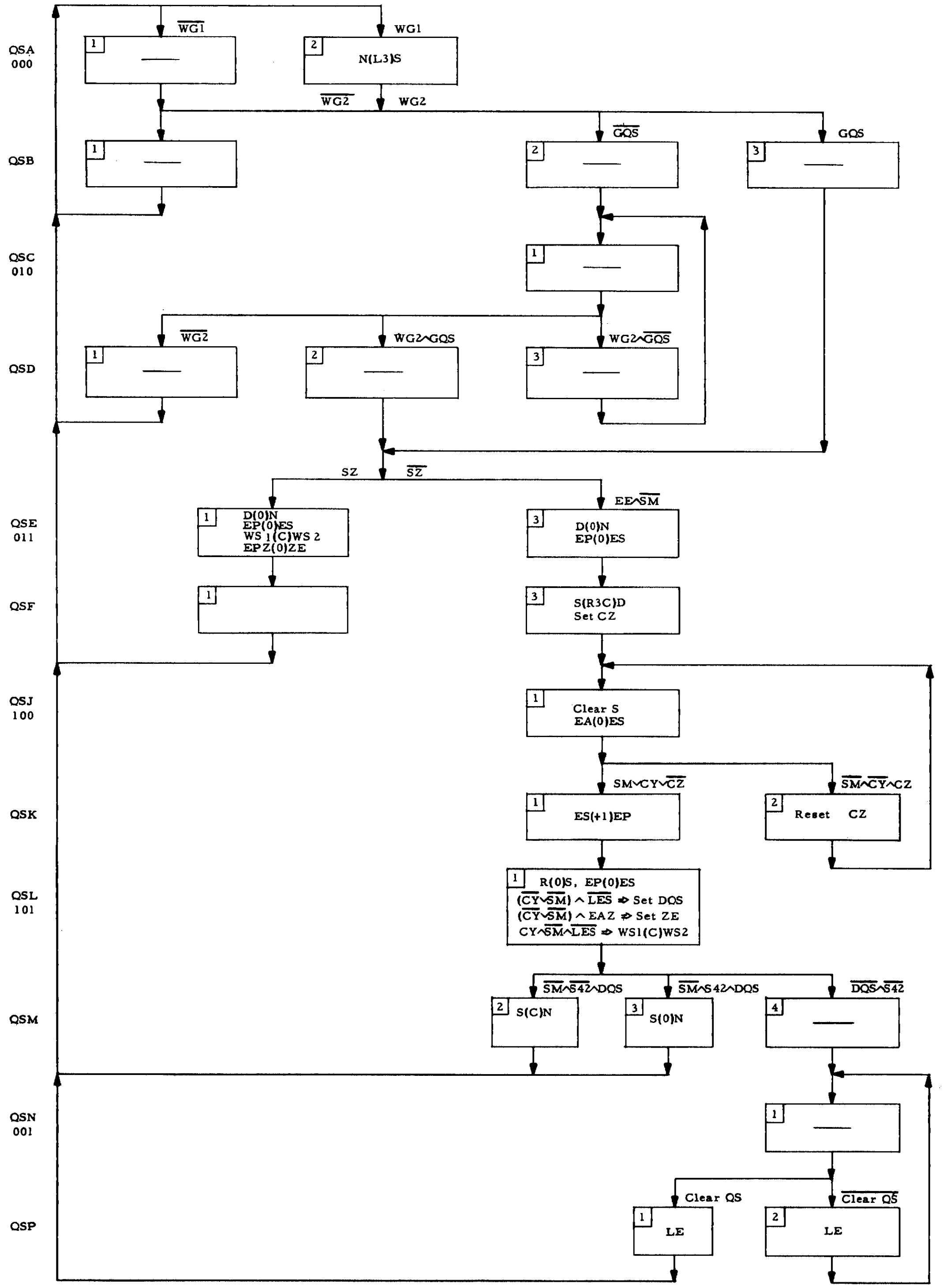

Idle
If WG1 signal, (N) left 3 to S in preparation for zero test
No
GO QS and WG2 signal?
Yes
SM reset to indicate differencing operation ; WS holds sign of operand in N
(D) to N as result; Reverse sign value ; Final exponent in EP to ES
Yes
(N) = 0?
No
Exponents equal?
No
See description of exponent equalization and Figures 13.6–6 and 13.6–8
Yes
(N) complemented, sent to D; (D) to N; Set carry–in to adder
Clear S for adder output; Final exponent in EA sent to ES
Reset carry–in flip–flop
Yes
Carry–in set and no carry–out?
No
Gate adder to S; (ES) plus 1 to EP
Idle until cleared by higher control
Yes
Exponent overflow condition?
No
Send DONE signal
(S) complemented, sent to N
No
Larger operand in D (Did carry–out occur?)
Yes
(S) to N; Reverse sign value
Result in register N; Final exponent in register EA ; Exponent value plus 1 in EP and ES; Final sign in SM and WS

TABLE 13.6–3 Terms Used on the QS Flow Chart for Differencing

Clear QS	Signal from higher level control. On detection of exponent overflow by QS, user sequencer requests an interrupt, clears QS, and takes any actions necessary to its own status. QS, on being sent a clear signal, returns to idle.
CY	CarrY inverter; CY high indicates that a mantissa overflow has occurred in the Adder output. CY, if enabled (CYE high), sets S42 or S43 depending on the output path from the Adder.
CYE	CY Enable flip–flop.
CZ	Carry Zero flip–flop; CZ high indicates a carry into the zero order of the Adder. CZ is set before differencing operations to effect the anticipated end around carry.
DQS	DONE QS sequencer.
EAZ	EA register Zero inverter; when high, EAZ indicates that the exponent held in register EA is zero.
EE	Exponents Equal signal; EE indicates (EA) = (EP).
EPZ	EP register Zero inverter; when high, indicates that the exponent held in register EP is zero.
GQS	GO signal to QS sequencer; GQS must occur when WG2 is high in order to start the sequencer.
LE	Large Exponent; LE indicates that an exponent overflow has occurred.
LES	Large Exponent Signal; LES is high when (EP) = 101_8. (This is 77_8 plus 1 since the exponent in EP is one greater than the final exponent when the result is gated from the Adder.)
R(0)S	Path enabled to gate output from Adder.
SM	SuM flip–flop; $\overline{SM}$ high indicates that a differencing operation is called for.
S42	Bit 42 in register S; if the output from the Adder is gated R(0)S, and CY $\wedge$ CYE are high, S42 goes high.
SZ	S register Zero signal; SZ is high when register S holds zero.
WG1, WG2	Wait Gate signals; WG2 is used in conjunction with GQS to start the QS sequencer.
WS	Working Sign flip–flop; WS holds the sign of the operand in N at GO QS and that of the result at DONE QS.
ZE	Zero Exponent signal; ZE comes high in QS when the final exponent is zero.

FIGURE 13.6-4 Paths Enabled During Exponent Equalization QS



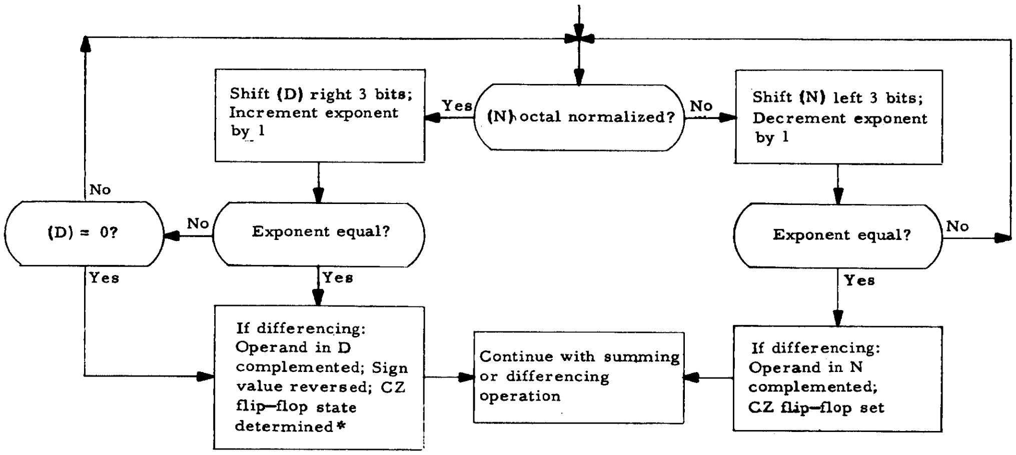
is switched with that in register D.) If (N) become octal normalized before equalization of exponents occurs, i.e., if significant bits are shifted into the high order octal position os register N, no further shifts are performed on that mantissa. Rather, equalization proceeds by means of shifting the mantissa in register D to the right with concurrent exponent increments. This process continues until either the exponents become equal or all the significant bits in register D are shifted out of the register. In the latter case the operand held in N and its exponent value are taken as the final result. On right shifts, loss of significant bits is remembered for use in rounding.

Another difference between the mode of operation of QS and that of QZ concerns the action taken when the exponents have become equal. In QZ, if the mantissas have been switched, they are returned to the original positions before control goes back to the higher level sequencers for completion of the operation. In QS, the mantissas are not exchanged but are gated through the Adder immediately. Control does not return to the higher level sequencer until the final result is available.

The basic algorithm for exponent equalization is shown in Figure 13.6-5. It should be noted that a test is made for (N) = 0 before the question of exponent equalization arises. Thus, it is only necessary to check for (D) = 0 during the equalization process. The algorithm shown is for the case where the mantissa associated with the larger exponent is stored in register N. Since this exponent is larger, it calls for exponent decrements and for left shifts of the associated mantissa. If (N) become octal normalized before exponent equalization occurs, the mantissa in register D is shifted right. Since the available shift paths for register N allow for left shifts only, while those for register D allow for right shifts, this is the simple case.

When the mantissa with the larger exponent is stored in register D, the mantissas must be interchanged in order to implement the left shifts required for the mantissa originally stored in D. It is not necessary to interchange the exponent values because both EA and EP have increment and decrement paths.

FIGURE 13.6–5 Basic Algorithm for Exponent Equalization



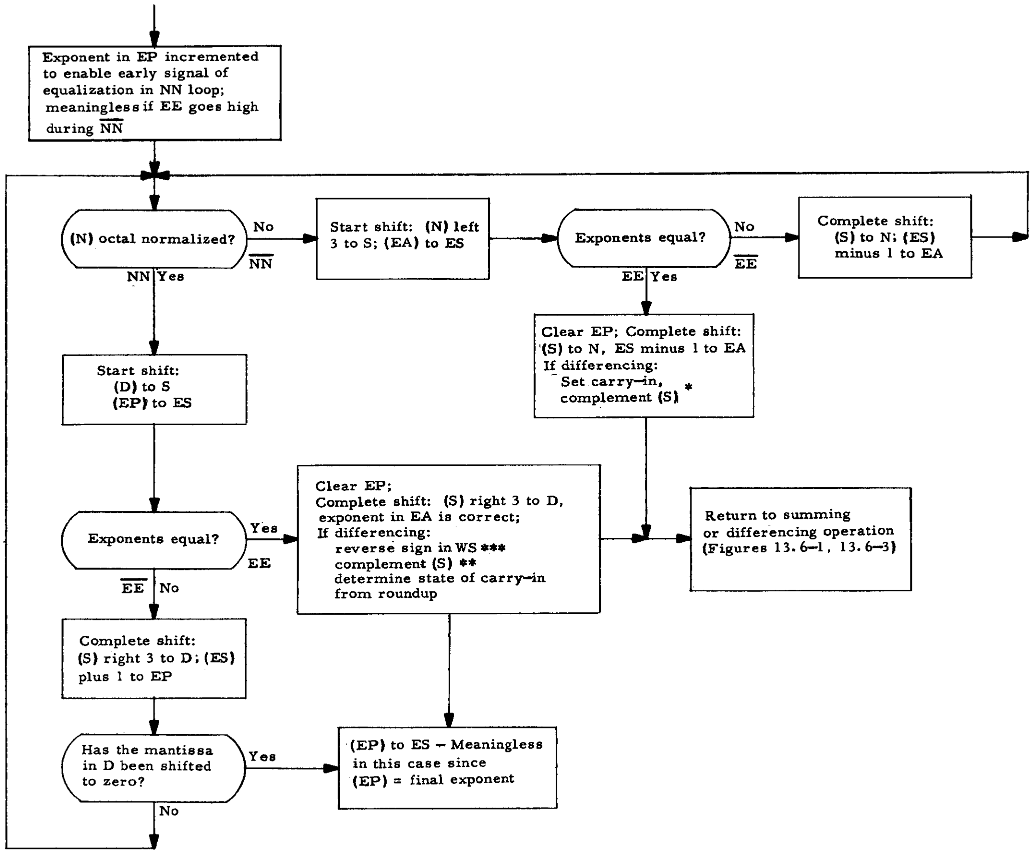
Shift (D) right 3 bits; Increment exponent by 1
Yes
(N) octal normalized?
No
Shift (N) left 3 bits; Decrement exponent by 1
No
(D) = 0?
No
Exponent equal?
Exponent equal?
No
Yes
Yes
Yes
If differencing: Operand in D complemented; Sign value reversed; CZ flip–flop state determined*
Continue with summing or differencing operation
If differencing: Operand in N complemented; CZ flip–flop set

*Basic differencing operation assumes /D/>/N/. The WS flip–flop holds the sign of (N), the operand that is complemented in assumed case. At DONE QS, if expected carry–out has occurred, verifying /D/>/N/, the sign in held in WS is reversed to give it the sign of (D) rather than (N). When exponent equalization takes place during the shifting of (D), the operand in D is complemented. This reverses the meaning of the carry–out signal. Hence, the sign held in WS is reversed.

It has been pointed out earlier that the CZ flip–flop is set prior to differencing operations. When (D) are shifted right, two factors influence the state of the CZ flip–flop: the state of the RU flip–flop (set when rounding is required due to loss of significant bits) and the requirements of the differencing operation. If CZ is assumed to be in the set condition due to the differencing operation, the occurrence of RU set will cause CZ to go low again. Since CZ is always set for differencing, the final state in this case is always the opposite of the RU flip–flop. Consequently, the complement of RU is sent to CZ for this case. On right shifts during summing operations the state of RU affects CZ directly.

The fact that the mantissas are not then stored in the registers

FIGURE 13.6–6 Algorithm for Equalizing Exponents when EL is High – (EA) > (EP)


Exponent in EP incremented to enable early signal of equalization in NN loop; meaningless if EE goes high during $\overline{NN}$
(N) octal normalized?
No
$\overline{NN}$
NN
Yes
Start shift: (N) left 3 to S; (EA) to ES
Exponents equal?
No
$\overline{EE}$
EE
Yes
Complete shift: (S) to N; (ES) minus 1 to EA
Clear EP; Complete shift: (S) to N, ES minus 1 to EA
If differencing: Set carry–in, complement (S) *
Start shift: (D) to S (EP) to ES
Exponents equal?
Yes
EE
$\overline{EE}$
No
Clear EP; Complete shift: (S) right 3 to D, exponent in EA is correct; If differencing: reverse sign in WS *** complement (S) ** determine state of carry–in from roundup
Return to summing or differencing operation (Figures 13.6–1, 13.6–3)
Complete shift: (S) right 3 to D; (ES) plus 1 to EP
Has the mantissa in D been shifted to zero?
Yes
No
(EP) to ES – Meaningless in this case since (EP) = final exponent

*Operand that was in register N at GO QS.

**Operand that was in register D at GO QS.

***Sign reversal accompanies complementation of operand that was in D at GQS since the basic algorithm assumes that the N operand will be complemented and that WS reflects the sign of the complemented operand.

associated with their respective exponent values does not affect the operation since, in shifting the mantissas in N or S, QS decrements or increments the correct exponent whether it is stored in EP or EA. Shifting continues until either the exponents are equal or the contents of D become zero so that, when the add cycle is entered, the exponents will either be equal or only one of them will be meaningful.

In Figure 13.6–6 the equalization algorithm shown is the detailed counterpart of the basic algorithm of Figure 13.6–5. For this case it is assumed that (EA) > (EP). This exponent relationship is indicated by the exponent larger signal, EL, which is high due to the exponent circuitry when (EA) > (EP). The paths used in implementing this algorithm are shown in Table 13.6–4.

TABLE 13.6–4 Paths Enabled During Exponent Equalization

(N) Not Octal Normalized		(N) Octal Normalized	
Mantissa shifted left:	N(L3)S S(0)N	Mantissa shifted right:	D(0)S S(R3)D
Exponent decremented:	EA(0)ES ES(−1)EA	Exponent incremented:	EP(0)ES ES(+1)EP

The algorithm is, for the most part, self–explanatory. The following points require some discussion.

1) (EP) are incremented before the shifting process begins, thereby putting the exponent associated with the (D) one step ahead of the mantissa shifts. (Since EL is high, the exponent stored in EP is the smaller one.) This increment takes place because the assumption is made that it will be necessary to shift (D) to the right before equalization is achieved. This early increment will, during right shifts, cause equalization to be signalled one shift early and, thus, makes it possible to complement, if required, on the final shift. When equalization occurs before (N) are octal normalized, a final shift and exponent decrement take place after equalization is signalled.

2) A meaningless transfer of (EP) into ES takes place when (D) are shifted to zero. The transfer occurs because it is necessary in the case where the exponent stored in EP is larger than that in EA (the case indicated by the signal $\overline{\text{EL}}$) and, therefore, the (N) and (D) have been interchanged. When this happens, and (D) are shifted to zero, the exponent of the result is that stored in EP and hence, the transfer is necessary. It remains in this algorithm since it does no harm and extra circuitry would be

involved in inhibiting it. The transfer will shortly be wiped out by the EA(0)ES transfer when the summing or differencing process continues (see Figures 13.6–1 and 13.6–3).

Figures 13.6–8 and 13.6–9 represent the $\overline{\text{EL}}$ case. This, of course, resembles what happens for EL, but some confusion can arise due to the mantissa interchange. (D) and (N) are switched by means of enabling three paths: N(L3)S, D(0)N, S(R3)D. It should be noted that (EP) are decremented while (N) are shifted left, and the (EA) are incremented when (D) are shifted right. The early increment this time involves the EA register. A further difference is that a test must again be made for (N) = 0 since the earlier test for this condition involved the mantissa that is now stored in register D.

TABLE 13.6–5 Terms Used on QS Flow Chart for Exponent Equalization

CZ	Carry Zero flip–flop; CZ is set to cause a carry into the zero order bit of the Adder. It is set on summing operations if the RU inverter is high; it is set before differencing operations unless right shifts of (D) were required and the RU signal is high.
DB	Discarded Bits flip–flop; DB is set when any of the 3 bits shifted out of register D on the right-3 shift is a 1; thus $DB \Rightarrow D{-}1 \vee D{-}2 \vee D{-}3$.
EE	Exponents Equal signal; $EE \Rightarrow (EA) = (EP)$.
EL	Exponents Larger flip–flop; EL is set when $(EA) > (EP)$; reset when $(EA) < (EP)$.
NN	Octal Normalized inverter; NN is high when the operand in register N becomes octal normalized, i.e., when there are significant bits in any of the 3 most significant bit positions so that no further shifts can be performed on this mantissa without loss of significant bits.
RU	Round–Up inverter; RU is set during right shifts in accordance with the rounding rule. $RU \Rightarrow S2 \wedge (S0 \vee S1 \vee \overline{S3} \vee DB)$.
SM	SuM flip–flop; SM is set for summing operations, reset for differencing.
SZ	S register Zero signal; SZ comes high in QS when register S holds zeros.
WS	Working Sign flip–flop; WS holds the sign of the operand in N at GO QS and that of the result at DONE QS. It is reversed during exponent equalization if the operand originally stored in register D is complemented.

FIGURE 13.6-7 The QS Flow Chart for Equalizing Exponents when EL is High – (EA) > (EP)

OS

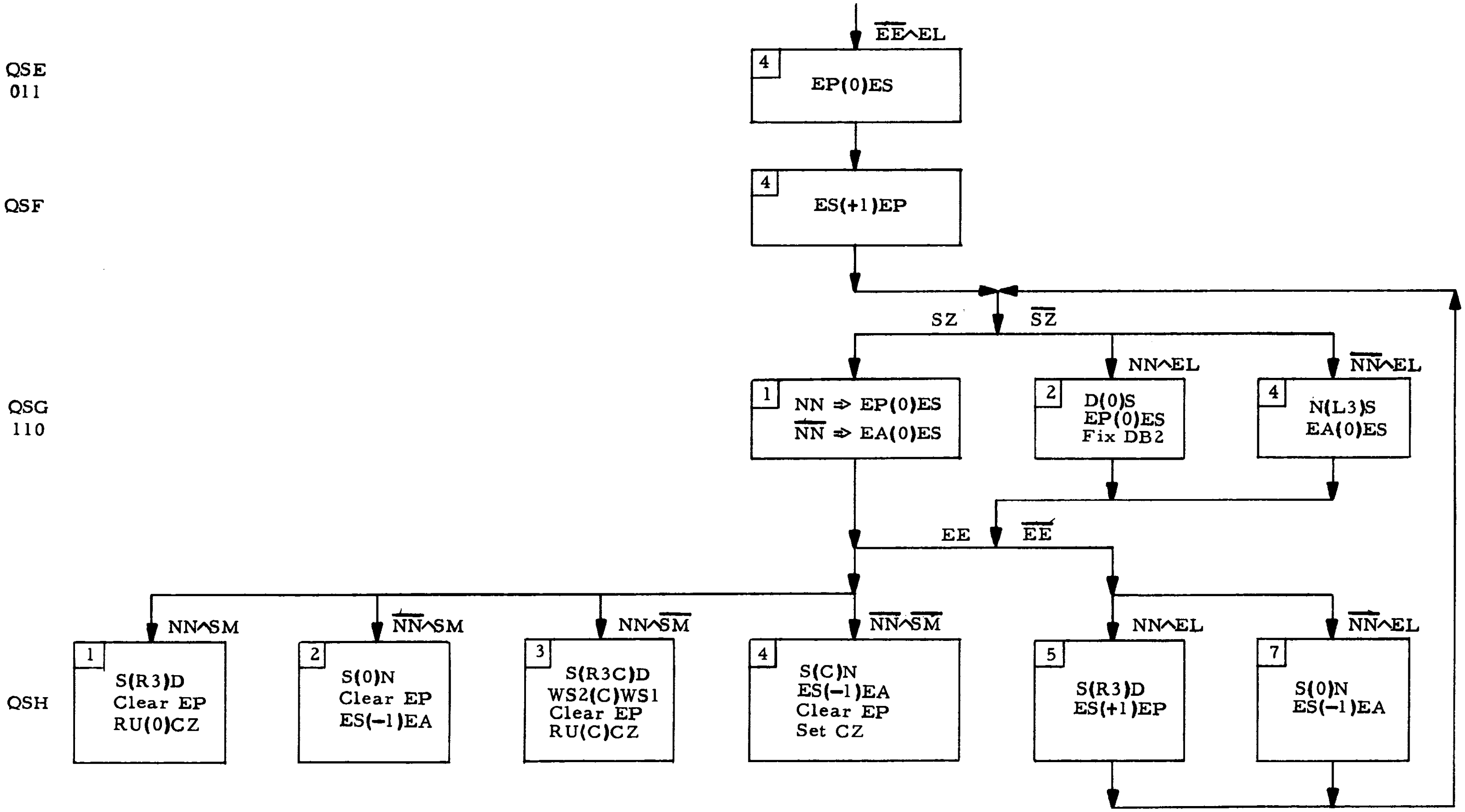
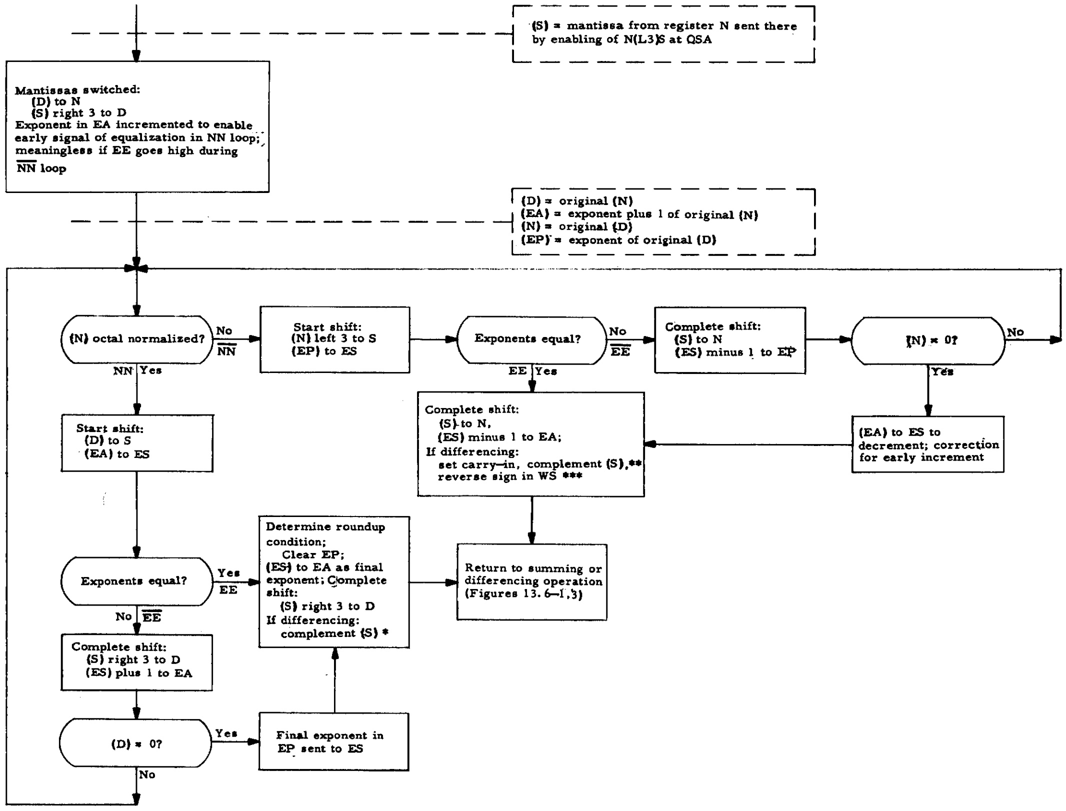


FIGURE 13.6–8 Algorithm for Equalizing Exponents when $\overline{\text{EL}}$ is High – (EA) < (EP)



*Operand in register N at GO QS.

**Operand in register D at GO QS.

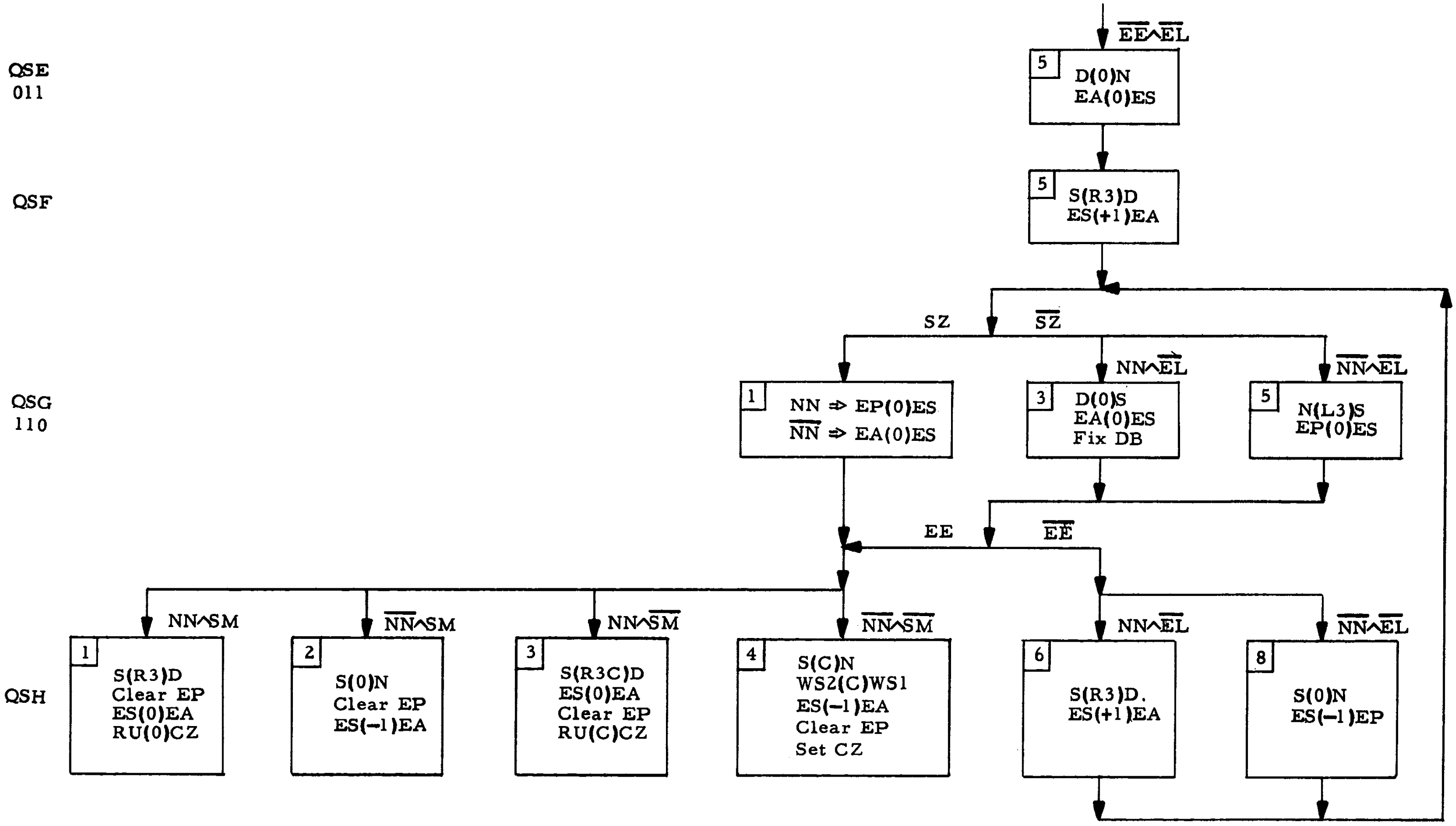
***Sign reversal accompanies complementation of operand that was in D at GO QS since the basic algorithm assumes that the N operand will be complemented and that WS reflects the sign of the complemented operand.

TABLE 13.6–6 Terms Used on QS Flow Chart for Exponent Equalization

CZ	Carry Zero flip–flop; CZ is set to cause a carry into the zero order bit of the Adder. It is set on summing operations if the RU inverter is high; it is set before differencing operations unless right shifts of (D) were required and the RU signal is high.
DB	Discarded Bits flip–flop; DB is set when any of the 3 bits shifted out of register D on a right–3 shift is a 1; thus $DB \Rightarrow D{-}1 \vee D{-}2 \vee D{-}3$.
EE	Exponents Equal signal; $EE \Rightarrow (EA) = (EP)$.
EL	Exponent Larger flip–flop; EL is set when $(EA) > (EP)$; reset when $(EA) < (EP)$.
NN	Octal Normalized inverter; NN is high when the operand in register N becomes octal normalized, i.e., when there are significant bits in any of the 3 most significant bit positions so that no further shifts can be performed on this mantissa without loss of significance.
RU	Round–Up inverter; RU is high during right shifts in accordance with the rounding rule. $RU \Rightarrow S2 \wedge (S0 \vee S1 \vee \overline{S3} \vee DB)$.
SM	SuM flip–flop; SM is set for summing operations, reset for differencing.
SZ	S register Zero signal; SZ comes high in QS when register S holds zeros.
WS	Working Sign flip–flop; WS holds the sign of the operand in N at GO QS and that of the result at DONE QS. It is reversed during exponent equalization if the operand originally stored in register D is complemented.

FIGURE 13.6–9 The QS Flow Chart for Equalizing Exponents when $\overline{EL}$ is High – (EA) < (EP)

QS



QSE 011
QSF
QSG 110
QSH
$\overline{EE} \wedge \overline{EL}$
5 D(0)N EA(0)ES
5 S(R3)D ES(+1)EA
SZ
$\overline{SZ}$
NN∧$\overline{EL}$
$\overline{NN} \wedge \overline{EL}$
1 NN ⇒ EP(0)ES $\overline{NN}$ ⇒ EA(0)ES
3 D(0)S EA(0)ES Fix DB
5 N(L3)S EP(0)ES
EE
$\overline{EE}$
NN∧SM
$\overline{NN}$∧SM
NN∧$\overline{SM}$
$\overline{NN} \wedge \overline{SM}$
NN∧$\overline{EL}$
$\overline{NN} \wedge \overline{EL}$
1 S(R3)D Clear EP ES(0)EA RU(0)CZ
2 S(0)N Clear EP ES(–1)EA
3 S(R3C)D ES(0)EA Clear EP RU(C)CZ
4 S(C)N WS2(C)WS1 ES(–1)EA Clear EP Set CZ
6 S(R3)D. ES(+1)EA
8 S(0)N ES(–1)EP

TABLE 13.6–7 Terms Used on the QS Flow Chart for Miscellaneous Operations Performed by QS

(These terms are general in nature and have, therefore, been omitted from earlier charts. They are mainly concerned with the bookkeeping operations necessary to the operation of QS.)

Clear EP	The final exponent is now in EA; register EP is left cleared for the next user.
Clear QS	This operation, which occurs in KA and KC, takes QS out of the idle loop it enters due to an exponent overflow and takes care of all necessary cleanup operations so that QS can be entered again.
CZ	Carry into Zero Adder bit flip–flop; CZ is assumed to be reset when QS is entered and must, hence, be reset before exiting. (See also discussion of differencing operation.)
DB	Discarded Bits flip–flop. DB is used external to QS and is reset here in case it has been set during operation of QS so that it is ready for the next user.
DQS	DONE QS flip–flop. DQS is set to indicate that QS has successfully completed its operation and is ready to exit. It is reset at QSA (when QS returns to idle) since the result of the last operation may no longer be available in OA.
QS8	QS8 flip–flop; this signal causes sending of an SKC or EARLY START signal to KC. QS8 is set when QS is far enough advanced in its operation that another command access can begin. QS8 is reset at QSL so that it will be in the correct state when QS is entered the next time.
ZE	Zero Exponent signal; ZE2 is reset at QSE, after QS is entered, but before ZE could be meaningful to the current operation, due to the fact that it is used by other sequencers and no assumptions can be made about its status at the time QS is entered.
WS(C)SM	This path is enabled each time QS completes an error–free operation, i.e., each time DQS is sent. The WS flip–flop, at the end of the operation, holds the sign of the result. This transfer leaves the final sign in the SM flip–flop. (WS states have the opposite meaning from those of SM; thus, $\overline{\text{WS}}$ and SM both indicate positive.)

FIGURE 13.6–10 Miscellaneous Operations Performed by QS

QS

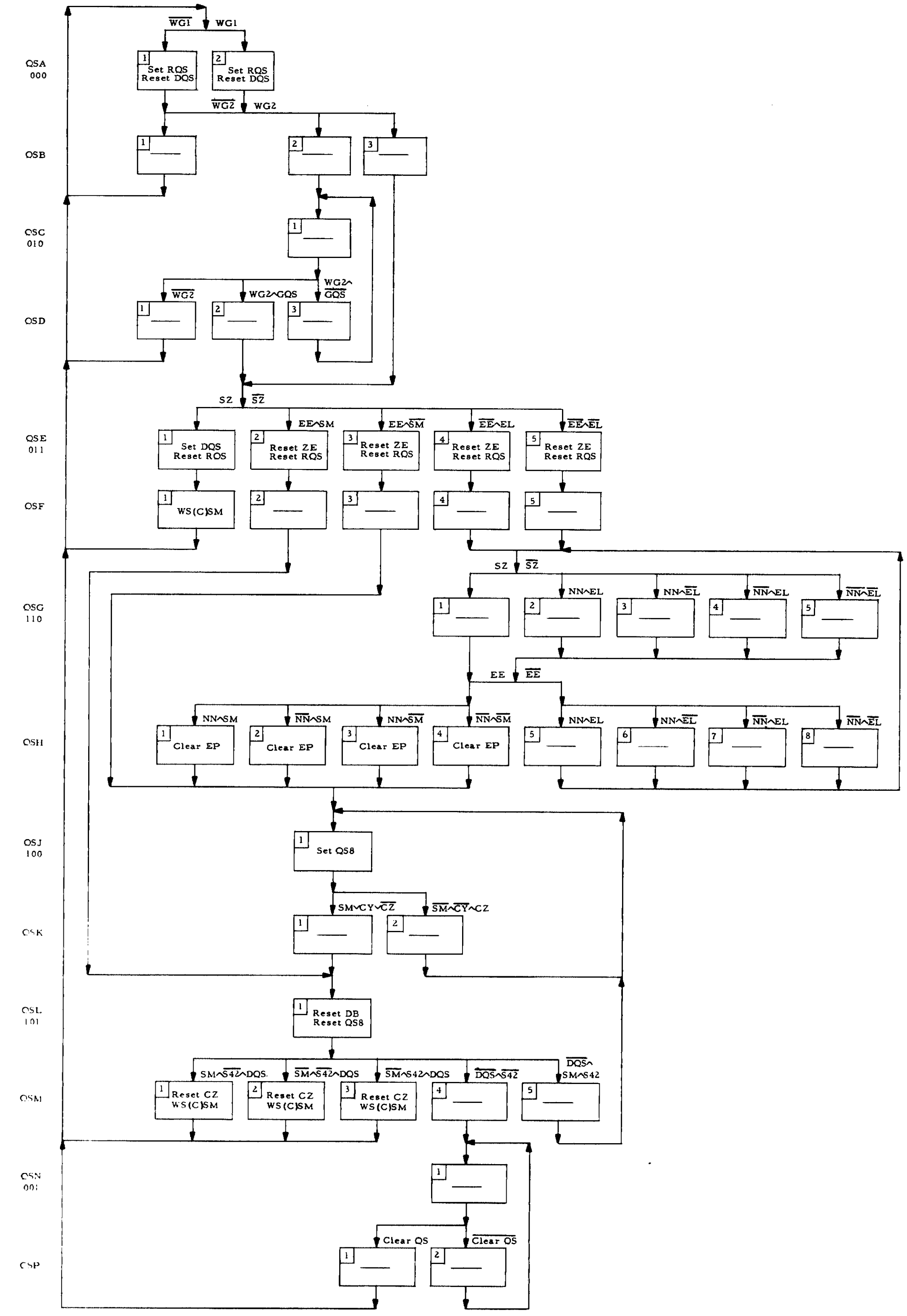
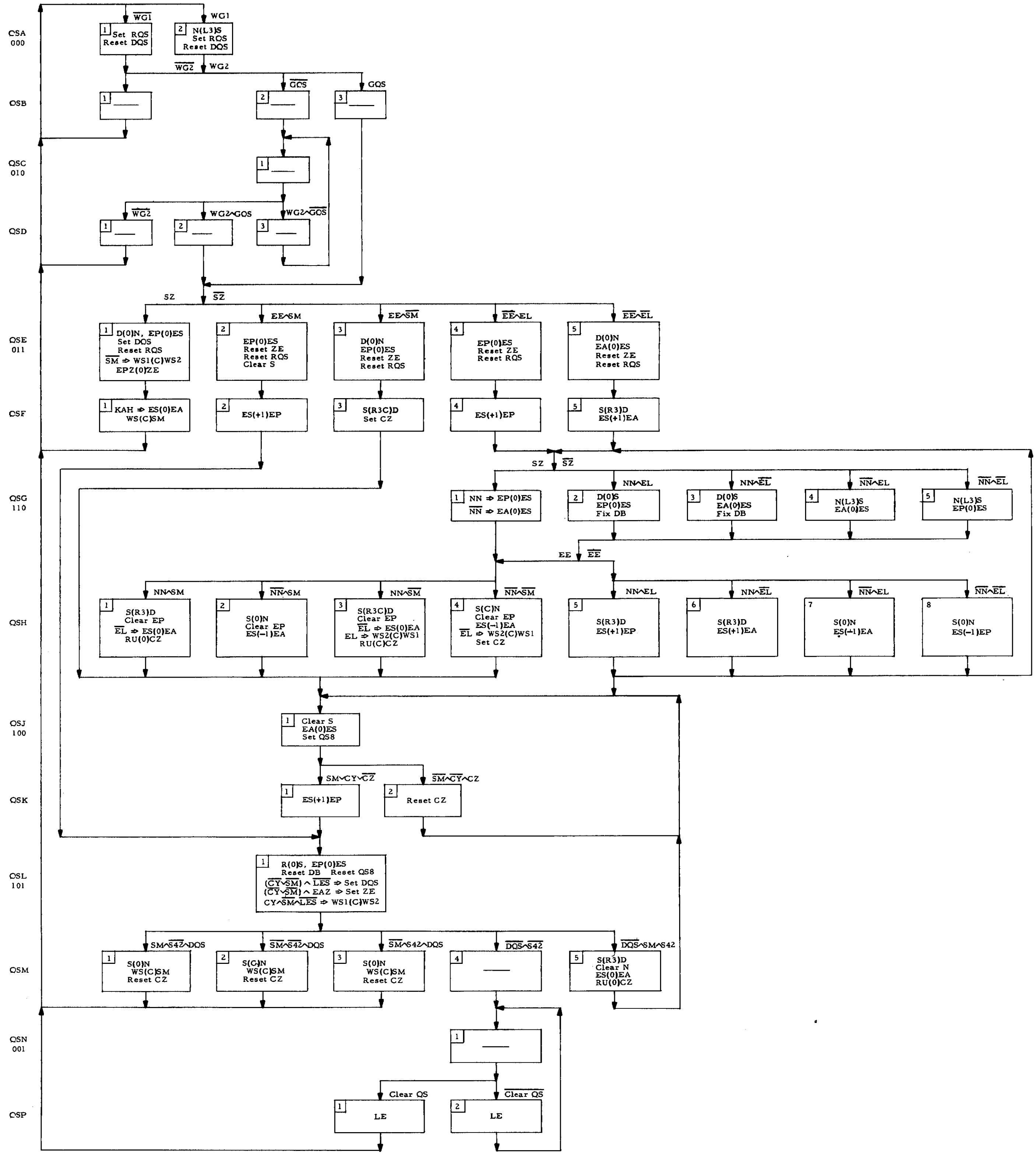


FIGURE 13.6–11 The QS Flow Chart

QS


QSA 000
QSB
QSC 010
QSD
QSE 011
QSF
QSG 110
QSH
QSJ 100
QSK
QSL 101
QSM
QSN 001
QSP
$\overline{WG1}$
WG1
1 Set RQS Reset DQS
2 N(L3)S Set RQS Reset DQS
$\overline{WG2}$
WG2
$\overline{GQS}$
GQS
1 ——
2 ——
3 ——
1 ——
$\overline{WG2}$
WG2∧GQS
WG2∧$\overline{GQS}$
1 ——
2 ——
3 ——
SZ
$\overline{SZ}$
EE∧SM
EE∧$\overline{SM}$
$\overline{EE}$∧EL
$\overline{EE}$∧$\overline{EL}$
1 D(0)N, EP(0)ES Set DQS Reset RQS $\overline{SM}$ ⇒ WS1(C)WS2 EPZ(0)ZE
2 EP(0)ES Reset ZE Reset RQS Clear S
3 D(0)N EP(0)ES Reset ZE Reset RQS
4 EP(0)ES Reset ZE Reset RQS
5 D(0)N EA(0)ES Reset ZE Reset RQS
1 KAH ⇒ ES(0)EA WS(C)SM
2 ES(+1)EP
3 S(R3C)D Set CZ
4 ES(+1)EP
5 S(R3)D ES(+1)EA
SZ
$\overline{SZ}$
NN∧EL
NN∧$\overline{EL}$
$\overline{NN}$∧EL
$\overline{NN}$∧$\overline{EL}$
1 NN ⇒ EP(0)ES $\overline{NN}$ ⇒ EA(0)ES
2 D(0)S EP(0)ES Fix DB
3 D(0)S EA(0)ES Fix DB
4 N(L3)S EA(0)ES
5 N(L3)S EP(0)ES
EE
$\overline{EE}$
NN∧SM
$\overline{NN}$∧SM
NN∧$\overline{SM}$
$\overline{NN}$∧$\overline{SM}$
NN∧EL
NN∧$\overline{EL}$
$\overline{NN}$∧EL
$\overline{NN}$∧$\overline{EL}$
1 S(R3)D Clear EP $\overline{EL}$ ⇒ ES(0)EA RU(0)CZ
2 S(0)N Clear EP ES(−1)EA
3 S(R3C)D Clear EP $\overline{EL}$ ⇒ ES(0)EA EL ⇒ WS2(C)WS1 RU(C)CZ
4 S(C)N Clear EP ES(−1)EA $\overline{EL}$ ⇒ WS2(C)WS1 Set CZ
5 S(R3)D ES(+1)EP
6 S(R3)D ES(+1)EA
7 S(0)N ES(−1)EA
8 S(0)N ES(−1)EP
1 Clear S EA(0)ES Set QS8
SM∨CY∨$\overline{CZ}$
$\overline{SM}$∧$\overline{CY}$∧CZ
1 ES(+1)EP
2 Reset CZ
1 R(0)S, EP(0)ES Reset DB Reset QS8 ($\overline{CY}$∨$\overline{SM}$) ∧ $\overline{LES}$ ⇒ Set DQS ($\overline{CY}$∨$\overline{SM}$) ∧ EAZ ⇒ Set ZE CY∧$\overline{SM}$∧$\overline{LES}$ ⇒ WS1(C)WS2
SM∧$\overline{S42}$∧DQS
$\overline{SM}$∧$\overline{S42}$∧DQS
$\overline{SM}$∧S42∧DQS
$\overline{DQS}$∧$\overline{S42}$
$\overline{DQS}$∧SM∧S42
1 S(0)N WS(C)SM Reset CZ
2 S(C)N WS(C)SM Reset CZ
3 S(0)N WS(C)SM Reset CZ
4 ——
5 S(R3)D Clear N ES(0)EA RU(0)CZ
1 ——
Clear QS
$\overline{Clear\ QS}$
1 LE
2 LE

SECTION 13.7 – THE KA SEQUENCER

KA controls the processing of the add/subtract opcodes gated through the KC sequencer in group 2f:

arithmetic operations A0, A1

arithmetic tests T0, T1

and group 2g:

arithmetic operations A2–7

arithmetic tests T2–7

logic operations L2, L3

logic tests S2, S3

address preparation operations N2–7

(Non–repeat use of commands A0 and A1, in modes 0 and 1, do not call for startup of KA; this case is handled by KC.) In execution of these opcodes, the QS sequencer is used to perform the required summing or differencing, QZ to shift the result to zero exponent for the L and S commands, and QA to store the result in the Accumulator for the A and L commands.

Two signals start the KA sequencer. These are called GKA and SKA despite the fact that the SKA signal does not exist. It has been used on the flow chart instead of RML, the equivalent existing signal, because of the general use of GO and START signals in sequencer terminology. This should cause no confusion as long as it is clear that the schematics refer to RML instead of SKA.

An SKA signal is sent to the KA sequencer at opcode startup for non–repeat operations in which one of the opcodes listed above has been written in mode 0 or 1. On receipt of an SKA signal, KA initializes the operation, starts QS to form the sum or difference, and, at DQS, completes the requirements of the opcode and informs KC. The exceptions to this are the T0 and T1 commands written in mode 0 or 1 which are processed by KA only because of the test requirements and which do not involve startup of QS.

A GKA signal is generated in two cases:

1) One of the commands listed above has been written in mode 2 or 3. When this occurs, KC sends the GKA signal to KA and initializes the operation before accessing the final operand. KA idles for DQS, completes the opcode requirements and returns control to KC.

2) One of the A, T, L or S commands listed above is written in the repeat mode. When this occurs, the KM sequencer (having been started by KC after decoding of the repeat command designation in the first command word) starts QC to access the second command word, initializes the operation, sends a GKA to KA, and signals KC to access the first operand in the block. KA idles until the DQS signal arrives. On subsequent operations, KA performs the initialization, signals KC to access the next operand, and goes to the idle loop where it waits for DQS.

A comparison of the two sources of the GKA signal is shown in Table 13.7–1. This distinction is made between the SKA and GKA cases because, on repeat as well as on mode 2 or 3 non–repeat commands, it is possible to effect a considerable saving in processing time by using the Master Control startup of QS at KCM (or KCR for double precision accesses) to form the sum or difference required by the current opcode. In the KCL loop of Master Control, the accessed information is sent to the D and EP registers and QS is started to add it to the partially assembled operand. In the case of GKA, no operand assembly is going on when this access is made. (For mode 2 or 3

TABLE 13.7–1 Comparison of KA Startups Through Use of Signal GKA

Repeat Commands (A, T, L2, L3, S2, S3)	Non–Repeat Commands (A, T, N, L2, L3, S2, S3) Written in Mode 2 or 3
1. KC assembles operand, starts KM, returns to idle. 2. KM starts QC to access second command word, initializes first sum or difference operation, sends GKA signal to advance KA to idle loop waiting for DQS, sends LKC signal to KC to start access of first operand, waits for signal of termination of repeat operation, returns control to KC and goes to idle. 3. KC on receipt of LKC: starts QM to access operand, idles for DAS signal, starts QS sequencer, gated to idle by term KA6 which is high when KA is in operation. 4. KA on receipt of GKA: idles for DQS, completes requirements of opcode, IF OPERATION NOT COMPLETE: initializes next sum or difference operation, sends LKC to KC to start access of next operand, returns to idle loop waiting for DQS signal. IF OPERATION COMPLETE: signals KM of termination, returns to normal idle.	1. KC assembles operand, sends operand to BA register for use as address in accessing final operand, starts QM to access final operand, initializes sum or difference operation, sends GKA signal to advance KA to idle loop waiting for DQS signal, waits for DAS signal, starts QS to perform sum or difference, gated to idle by term KA6 which is high when KA is in operation. 2. KA on receipt of GKA: idles for DQS, completes requirements of opcode, returns to normal idle.

commands, this is the final operand access using the assembled operand as an address; for repeat commands, no operand assembly process is involved.) Thus, it is possible to set up the second operand, if any, in OA and to determine the state of the SM flip–flop before the GQS signal is sent, thereby taking advantage of this startup of QS. The set–up process, which is referred to as an initialization, has been described to some extent in Section 13.1 in conjunction with the discussion of inputs to the QS sequencer. It will be reviewed here in detail in order to clarify the manner of implementation.

Each operation performed by KA requires initialization prior to the startup of QS. (Note that QS is not started for the commands T0 and T1 written in mode 0 or 1.) When an SKA signal starts KA, the initialization and the QS start are handled by KA. For the GKA branch, there are two possible controls:

1) for non–repeat commands written in mode 2 or 3, initialization occurs at KZD (or KZH for negative–zero addresses) in the KC sequencer;

2) for repeat operations (first iteration), initialization occurs at KMD in the KM sequencer.

Initialization is occurring when KA is advanced to state KAB. The actions involved are included on the KA flow chart at KAB for easy reference. KA then jumps to KAG–KAH where it idles for DQS. On subsequent iterations during repeat, KA completes initialization of each operation at KAP, then goes directly to the idle loop at KAG–KAH to await the DQS signal.

The term initialization covers the manipulations necessary to setting up the inputs to the QS sequencer. These are:

1) determination of the state of the SM flip–flop;

2) determination of the state of the WS flip-flop;

3) for single-operand commands registers N and EA are cleared and flip-flop WS is complemented if the opcode is odd-numbered;

4) for two-operand commands, the second operand is transferred from the Accumulator to OA. On SKA, the assembled operand is transferred from OA to D and EP. On GKA, the accessed operand goes directly to D and EP.

All sign manipulations are in accordance with the values shown in Table 13.1-2. At SKA, the assembled operand is in N, its exponent in the EA and EP registers, and the sign in the WS and SM flip-flops. In handling the single-operand commands gated by the term 2f, KC reverses the WS flip-flop for the odd-numbered commands. For A0 and A1, KA is not started up in mode 0 and 1. Rather, a start signal is sent to QA to store the operand in the Accumulator following the sign reversal if the command is A1. For T0 and T1, KA is started to carry out the test; QS is not used for these commands.

For the two operand commands gated through Master Control by the term 2g, the SKA signal finds the operands set up as follows:

assembled operand: N, EA and EP, WS, SM;
second operand: A, AE, AS.

The assembled operand is sent to D; its exponent is already in EP. The second operand is transferred to N and EA. The sign in AS is copied into WS (so that, during operation of QS, WS reflects the sign of the operand in register N). At this point, SM holds the sign of the assembled operand. If the opcode and the sign in AS are both positive or both negative, no change is required in the state of SM (see Table 13.1-2). On the other hand, if the signs are opposite one another, the state of SM is reversed. The term "Fix SM" on the KA flow chart refers to implementation of this rule via decoding of $\wedge \overline{CD10}$ (even-numbered opcodes) $\wedge AS \Rightarrow SM2(C)SM1$, or CD10 (odd-numbered opcodes) $\wedge \overline{AS} \Rightarrow$ SM2(C)SM1.

For the GKA case, two sign values are involved in the single-operand commands: that of the operand and that of the opcode. Since the operand from the Accumulator would normally be sent to OA before the summing or differencing operation, OA is cleared (registers N and EA cleared, the WS flip-flop reset). The SM flip-flop, which indicates a summing or differencing operation, is reset for the odd-numbered or subtract opcodes, and set for the even-numbered opcodes. (If the accessed operand is negative, indicated by B28 high, the state of SM is reversed at KCM.)

For the two operand commands, SM is initially adjusted to reflect the sign of the second operand combined with that of the operation itself. The sign of the accessed operand is the last to be taken into account since it is not available until the access takes place (just prior to the startup of QS). At GKA, the operands are stored as follows:

final operand: not yet available; access is just about to begin;

second operand: A, AE, AS.

The values in A and AE are sent to N and EA. Sign adjustments are made in order given below:

1) At KMD (repeat) or KZD (mode 2 or 3 non-repeat), the sign held in the AS flip-flop is sent to the WS flip-flop.

2) The GKA signal allows KA to do some decoding at state KAB where the opcode sign is taken into account. The value held in SM at this time is not relevant to the current operation since it reflects the assembly of the last operand. In the operand assembly process, SM holds the same sign as WS when the KCL section of KC is entered, but operand assembly calls only for addition whereas commands handled by KA call for addition or subtraction depending upon whether the command is odd or even

numbered. Since the approach taken is to artificially create the situation that would occur during operand assembly given these values so that a normal access will be handled correctly, it is necessary to adjust SM so that it reflects the combined sign of the operation (positive for addition, or even–numbered opcodes) and the sign of the operand from the Accumulator (the sign in AS which is also being copied into WS). Referring to Table 13.1–2, these values can be related to those given if the sign in B28 (thus far unknown since the operand has not been accessed) is assumed to be positive. The sign in AS takes the place of the sign value given for WS and SM for this determination since the operand from the Accumulator is assuming the role of the assembled operand. With B28 positive, the sign in SM should be the same as that in AS for the positive (even–numbered) opcodes. Since SM is in the opposite state from the other sign flip–flops to represent the same value, the state of SM is established by means of enabling the AS(C)SM path for the even–numbered opcodes. Similarly, if the opcode is odd–numbered, the state of SM is reversed by the enabling of the AS(0)SM path.

3) At KCM the accessed operand is available in the B register. If B28 is high, indicating a negative value, the SM flip–flop is reversed via the enabling of the SM2(C)SM1 path.

4) On subsequent repeat operations, KA handles 1) and 2) above at the KAP state. 3) occurs in the same way in all cases.

The basic processing distinctions made by KA are between the repeat and non–repeat cases and the test and non–test cases. Thus, the sequencer has been divided into four basic algorithms:

1) non–repeat logic, arithmetic and address preparation opcodes; (SKA for mode 0 or 1, GKA for mode 2 or 3); see Figures 13.7–1

and 13.7–2;

2) repeat logic and arithmetic opcodes; (address preparation commands are not available in the repeat mode); (these commands are started by a GKA from the KM sequencer); see Figures 13.7–3 and 13.7–4;

3) non–repeat arithmetic and logic tests; see Figures 13.7–5 and 13.7–6; (started by SKA for mode 0 or 1, GKA for mode 2 or 3);

4) repeat arithmetic and logic tests; (started by GKA from the KM sequencer); see Figures 13.7–7 and 13.7–8.

The manipulations described in these diagrams are based on the material in Sections 13.1 and 13.2 which give the basic rules for use of QS, the handling of repeat operations, the use of tests, etc. The behavior of the QS and KM sequencers is also relevant (Sections 13.3 and 13.6). To avoid redundancy, this information is not repeated in this section.

FIGURE 13.7–1 Algorithm for the Processing of Non–Repeat Logic, Arithmetic and Address Preparation Opcodes by KA

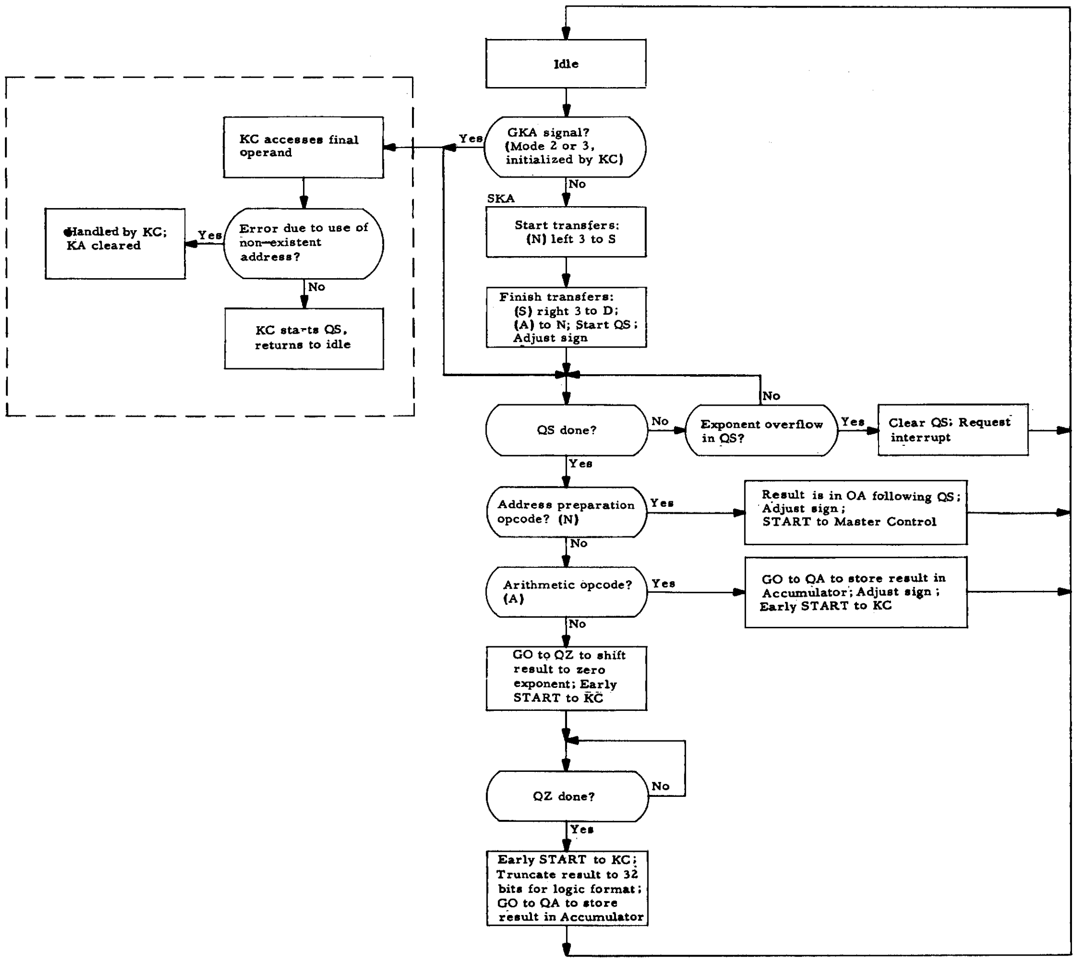

Idle
GKA signal? (Mode 2 or 3, initialized by KC)
Yes
No
KC accesses final operand
Error due to use of non–existent address?
Yes
Handled by KC; KA cleared
No
KC starts QS, returns to idle
SKA
Start transfers: (N) left 3 to S
Finish transfers: (S) right 3 to D; (A) to N; Start QS; Adjust sign
No
QS done?
No
Exponent overflow in QS?
Yes
Clear QS; Request interrupt
Yes
Address preparation opcode? (N)
Yes
Result is in OA following QS; Adjust sign; START to Master Control
No
Arithmetic opcode? (A)
Yes
GO to QA to store result in Accumulator; Adjust sign; Early START to KC
No
GO to QZ to shift result to zero exponent; Early START to KC
QZ done?
No
Yes
Early START to KC; Truncate result to 32 bits for logic format; GO to QA to store result in Accumulator

TABLE 13.7-2 Terms Used on the KA Flow Chart for the Processing of Non-Repeat Logic, Arithmetic and Address Preparation Opcodes

A0-7	Arithmetic commands.
AS	Accumulator Sign flip-flop; AS holds the sign of the operand stored in the Accumulator; AS high indicates negative sign.
DQS	DONE signal from the QS sequencer.
DRA	Done Repeat Arithmetic operation; DRA is sent to KM at the termination of repeat operations for L2, L3, A0-7 commands.
EVEN	Even-numbered opcodes.
Fix SM	SM is adjusted according to the rules described in Table 13.1-2.
Group 2e	Term used to gate commands N2-7, A2-7, T2-7, L2, L3, S2 and S3 through KC.
Group 2f	Term used to gate commands A0, A1, T0 and T1 through KC.
GKA	GO signal to KA sequencer.
KR	Opcode DONE signal to KC.
L2, L3	Logic commands ADL and SUL which are processed by KA.
LE	Large Exponent; LE is equivalent to state QSP of the QS sequencer.
MNA	Memory Non-existent Address signal; MNA is sent by QM on detection of a non-legal address during access.
N2-7	Address Preparation commands.
ODD	Odd-numbered opcodes.
Set JNC	Request interrupt due to use of non-existent address.
SKA	Start KA signal from KC; this signal is non-existent; it is equivalent to RML and indicates that the current command is non-repeat and has been written in mode 0 or 1.
SKC	Start KC; early Go to Master Control which places KC in the KCC state.
SM	SuM flip-flop; SM is set to direct QS to perform a summing operation,reset for differencing.
TKC	Tilt KC; TKC => CD8, Set KR1, Reset LP, thus enabling interrupts and disabling logic accesses. This exit is taken when KA is halted by the occurrence of a non-legal memory address.
WG1	Wait Gate signal; WG1 is sent to the idle state of QS to enable the N(L3)S path.
WG2	Wait Gate signal; WG2 is sent to QS along with GQS to start the QS sequencer. WG2 is not the echo signal of WG1.
WS	Working Sign flip-flop; WS holds the sign of the operand currently held in OA.
ZE	Zero Exponent flip-flop; ZE is set by KA when the EA exponent register contains zeros.

FIGURE 13.7–2 The KA Flow Chart for Processing Non–Repeat Logic, Arithmetic and Address Preparation Opcodes

KA

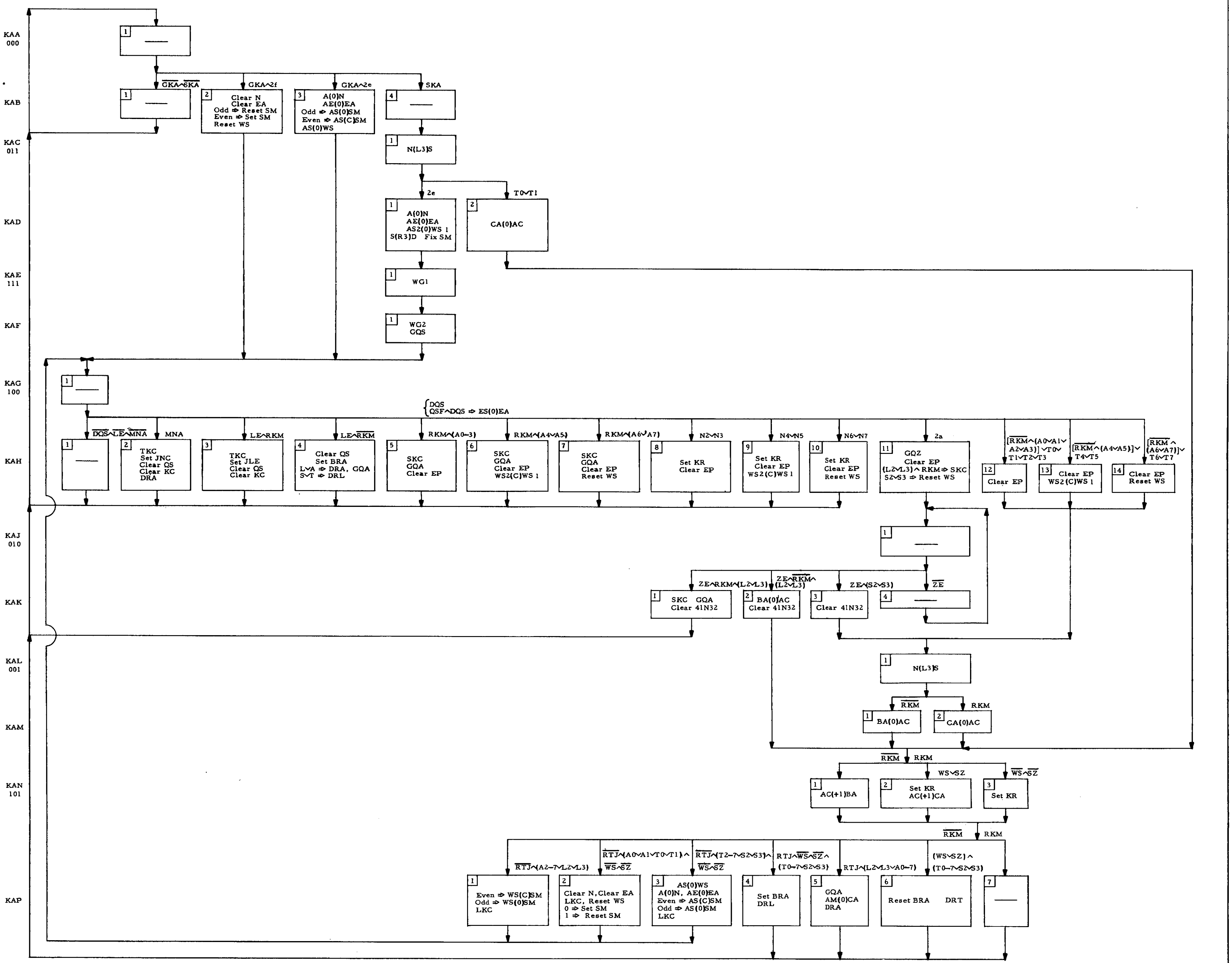

KAA 000
KAB
KAC 011
KAD
KAE 111
KAF
KAG 100
KAH
KAJ 010
KAK
KAL 001
KAM
KAN 101
KAP
$\overline{GKA}\wedge\overline{SKA}$
GKA∧2f
GKA∧2e
SKA
Clear N Clear EA Odd ⇒ Reset SM Even ⇒ Set SM Reset WS
A(0)N AE(0)EA Odd ⇒ AS(0)SM Even ⇒ AS(C)SM AS(0)WS
N(L3)S
2e
T0∨T1
A(0)N AE(0)EA AS2(0)WS 1 S(R3)D Fix SM
CA(0)AC
WG1
WG2 GQS
DQS
QSF∧DQS ⇒ ES(0)EA
$\overline{DQS}\wedge\overline{LE}\wedge\overline{MNA}$
MNA
LE∧RKM
$LE\wedge\overline{RKM}$
RKM∧(A0–3)
RKM∧(A4∨A5)
RKM∧(A6∨A7)
N2∨N3
N4∨N5
N6∨N7
2a
$[\overline{RKM}\wedge(A0\vee A1\vee A2\vee A3)]\vee T0\vee T1\vee T2\vee T3$
$[\overline{RKM}\wedge(A4\vee A5)]\vee T4\vee T5$
$[\overline{RKM}\wedge(A6\vee A7)]\vee T6\vee T7$
TKC Set JNC Clear QS Clear KC DRA
TKC Set JLE Clear QS Clear KC
Clear QS Set BRA L∨A ⇒ DRA, GQA S∨T ⇒ DRL
SKC GQA Clear EP
SKC GQA Clear EP WS2(C)WS 1
SKC GQA Clear EP Reset WS
Set KR Clear EP
Set KR Clear EP WS2(C)WS 1
Set KR Clear EP Reset WS
GQZ Clear EP (L2∨L3)∧RKM ⇒ SKC S2∨S3 ⇒ Reset WS
Clear EP
Clear EP WS2(C)WS 1
Clear EP Reset WS
ZE∧RKM∧(L2∨L3)
$ZE\wedge\overline{RKM}\wedge(L2\vee L3)$
ZE∧(S2∨S3)
$\overline{ZE}$
SKC GQA Clear 41N32
BA(0)AC Clear 41N32
Clear 41N32
N(L3)S
$\overline{RKM}$
RKM
BA(0)AC
CA(0)AC
$\overline{RKM}$
RKM
WS∨SZ
$\overline{WS}\wedge\overline{SZ}$
AC(+1)BA
Set KR AC(+1)CA
Set KR
$\overline{RKM}$
RKM
$\overline{RTJ}\wedge(A2–7\vee L2\vee L3)$
$\overline{RTJ}\wedge(A0\vee A1\vee T0\vee T1)\wedge\overline{WS}\wedge\overline{SZ}$
$\overline{RTJ}\wedge(T2–7\vee S2\vee S3)\wedge\overline{WS}\wedge\overline{SZ}$
$RTJ\wedge\overline{WS}\wedge\overline{SZ}\wedge(T0–7\vee S2\vee S3)$
RTJ∧(L2∨L3∨A0–7)
(WS∨SZ)∧(T0–7∨S2∨S3)
Even ⇒ WS(C)SM Odd ⇒ WS(0)SM LKC
Clear N, Clear EA LKC, Reset WS 0 ⇒ Set SM 1 ⇒ Reset SM
AS(0)WS A(0)N, AE(0)EA Even ⇒ AS(C)SM Odd ⇒ AS(0)SM LKC
Set BRA DRL
GQA AM(0)CA DRA
Reset BRA DRT

FIGURE 13.7–3 Algorithm for the Processing of Repeat Logic and Arithmetic Opcodes by KA

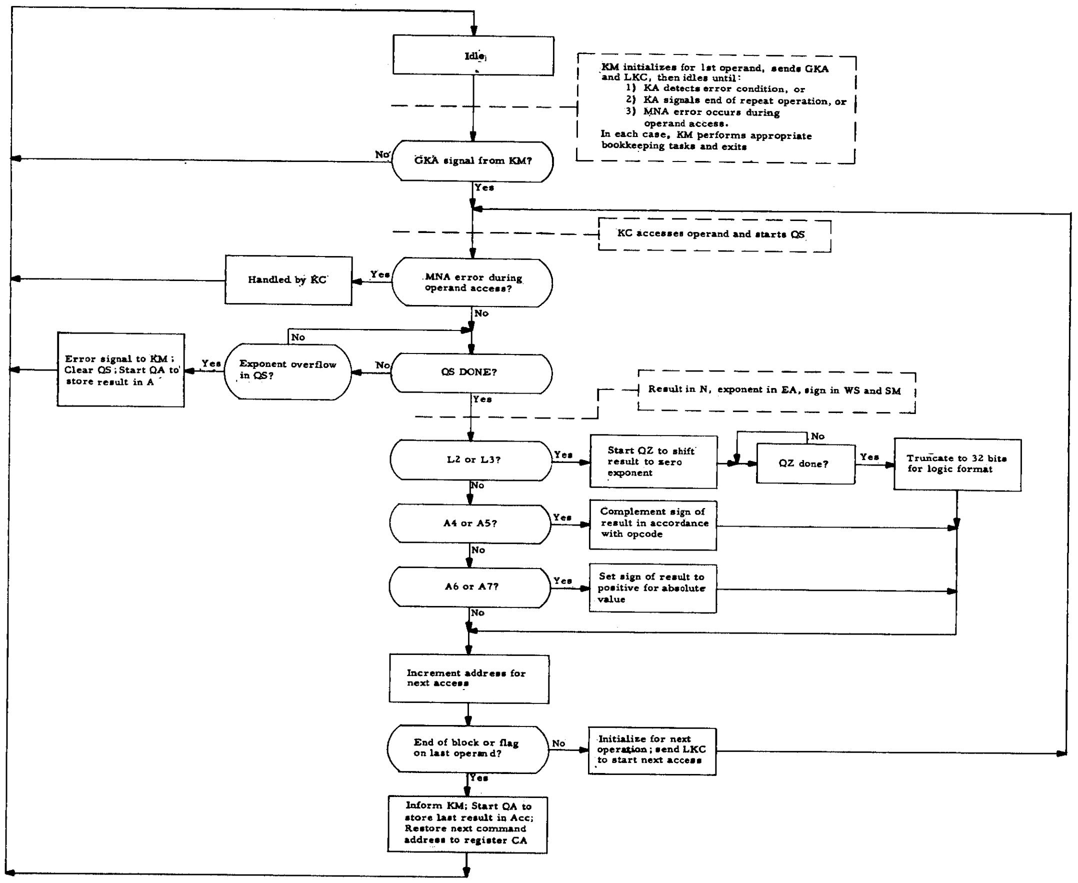

Idle
KM initializes for 1st operand, sends GKA and LKC, then idles until:
1) KA detects error condition, or
2) KA signals end of repeat operation, or
3) MNA error occurs during operand access.
In each case, KM performs appropriate bookkeeping tasks and exits
No
GKA signal from KM?
Yes
KC accesses operand and starts QS
Handled by KC
Yes
MNA error during operand access?
No
No
Error signal to KM; Clear QS; Start QA to store result in A
Yes
Exponent overflow in QS?
No
QS DONE?
Result in N, exponent in EA, sign in WS and SM
Yes
L2 or L3?
Yes
Start QZ to shift result to zero exponent
No
QZ done?
Yes
Truncate to 32 bits for logic format
No
A4 or A5?
Yes
Complement sign of result in accordance with opcode
No
A6 or A7?
Yes
Set sign of result to positive for absolute value
No
Increment address for next access
End of block or flag on last operand?
No
Initialize for next operation; send LKC to start next access
Yes
Inform KM; Start QA to store last result in Acc; Restore next command address to register CA

TABLE 13.7–3 Terms Used on the KA Flow Chart for Processing of Repeat Logic and Arithmetic Opcodes

A0–7	Arithmetic commands.
DQS	DONE signal from the QS sequencer.
DRA	Done Repeat Arithmetic operation; DRA is sent to KM at the termination of repeat operations for L2, L3, A0–7 commands.
EVEN	Even–numbered opcodes.
Group 2a	Term used to gate commands L2, L3, S2, S3 through KC.
Group 2e	Term used to gate commands N2–7, A2–7, T0–7, L2, L3, S2, and S3 through KC.
Group 2f	Term used to gate commands A0, A1, T0 and T1 through KC.
GKA	GO signal to KA sequencer.
L2, L3	Logic commands ADL and SUL which are processed by KA.
LE	Large Exponent; LE is equivalent to state QSP of the QS sequencer.
LKC	Loop KC sequencer; when high, LKC directs KC to access the next operand in repeat mode.
ODD	Odd–numbered opcodes.
QSF ^ DQS	Decoding indicating that the (N) = 0 and QS has taken an early exit path. Under these conditions, the path ES(0)EA is enabled to leave the exponent properly positioned in EA.
RTJ	Repeat Termination signal; RTJ is high when the end of the block is reached or an enabled flag (logic for L2, L3, S2, S3 commands, data for A or T commands) is detected.
SM	SuM flip–flop; SM is set to direct QS to perform a summing operation, reset for differencing.
SZ	S register Zero signal; SZ is high when register S contains zeros.
TKC	Tilt KC; TKC ⇒ Set CD8, Set KR1, Reset LP, thus enabling interrupts and disabling logic accesses. This exit is taken when KA is halted by the occurrence of a non–legal memory address.
WS	Working Sign flip–flop; WS holds the sign of the operand currently held in OA.
ZE	Zero Exponent signal; ZE comes high in KA when the EA exponent register contains zeros.

FIGURE 13.7–4 The KA Flow Chart for the Processing of Repeat Logic and Arithmetic Opcodes by KA

KA

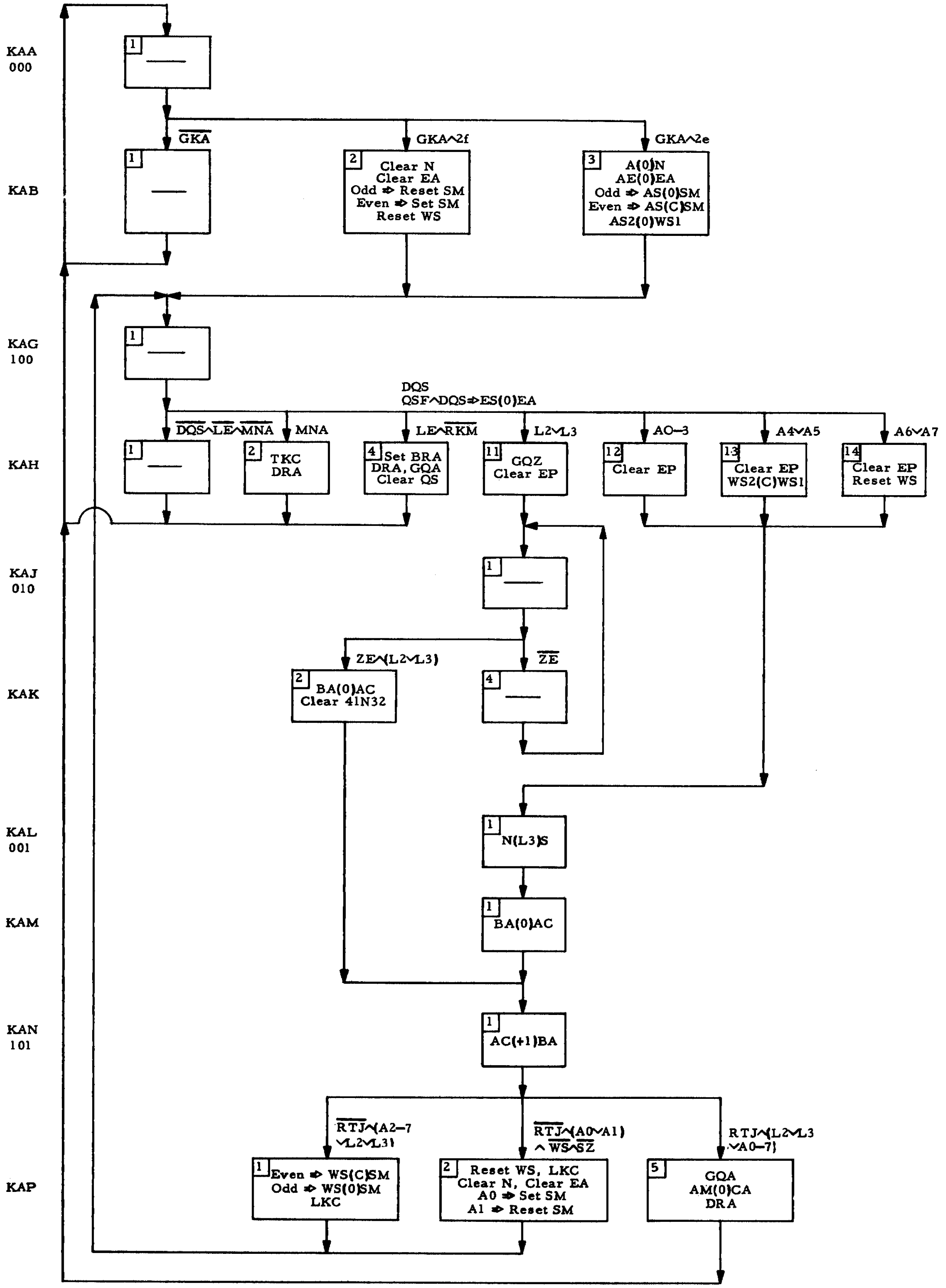

KAA 000
1 —
$\overline{GKA}$
GKA∧2f
GKA∧2e
KAB
1 —
2 Clear N Clear EA Odd ⇒ Reset SM Even ⇒ Set SM Reset WS
3 A(0)N AE(0)EA Odd ⇒ AS(0)SM Even ⇒ AS(C)SM AS2(0)WS1
KAG 100
1 —
DQS
QSF∧DQS⇒ES(0)EA
$\overline{DQS}$∧$\overline{LE}$∧$\overline{MNA}$
MNA
LE∧$\overline{RKM}$
L2∨L3
AO–3
A4∨A5
A6∨A7
KAH
1 —
2 TKC DRA
4 Set BRA DRA, GQA Clear QS
11 GQZ Clear EP
12 Clear EP
13 Clear EP WS2(C)WS1
14 Clear EP Reset WS
KAJ 010
1 —
ZE∧(L2∨L3)
$\overline{ZE}$
KAK
2 BA(0)AC Clear 41N32
4 —
KAL 001
1 N(L3)S
KAM
1 BA(0)AC
KAN 101
1 AC(+1)BA
$\overline{RTJ}$∧(A2–7 ∨L2∨L3)
$\overline{RTJ}$∧(A0∨A1) ∧ $\overline{WS}$∧$\overline{SZ}$
RTJ∧(L2∨L3 ∨A0–7)
KAP
1 Even ⇒ WS(C)SM Odd ⇒ WS(0)SM LKC
2 Reset WS, LKC Clear N, Clear EA A0 ⇒ Set SM A1 ⇒ Reset SM
5 GQA AM(0)CA DRA

FIGURE 13.7–5 Algorithm for the Processing of Non–Repeat Logic and Arithmetic Tests Handled by KA

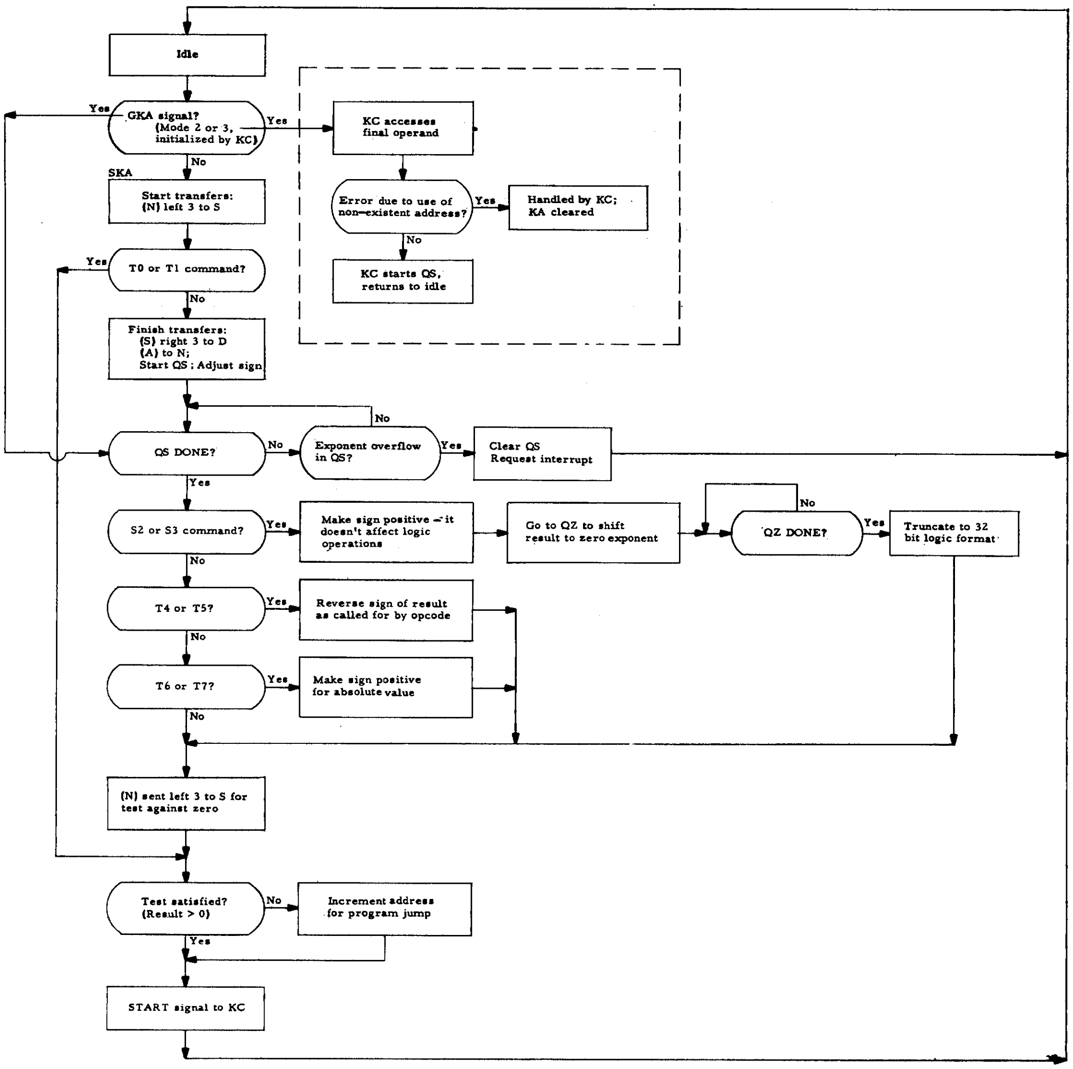

Idle
Yes
GKA signal? (Mode 2 or 3, initialized by KC)
Yes
No
KC accesses final operand
Error due to use of non–existent address?
Yes
Handled by KC; KA cleared
No
KC starts QS, returns to idle
SKA
Start transfers: (N) left 3 to S
Yes
T0 or T1 command?
No
Finish transfers: (S) right 3 to D (A) to N; Start QS ; Adjust sign
No
QS DONE?
No
Exponent overflow in QS?
Yes
Clear QS Request interrupt
Yes
S2 or S3 command?
Yes
Make sign positive – it doesn't affect logic operations
Go to QZ to shift result to zero exponent
No
QZ DONE?
Yes
Truncate to 32 bit logic format
No
T4 or T5?
Yes
Reverse sign of result as called for by opcode
No
T6 or T7?
Yes
Make sign positive for absolute value
No
(N) sent left 3 to S for test against zero
Test satisfied? (Result > 0)
No
Increment address for program jump
Yes
START signal to KC

TABLE 13.7–4 Terms Used on the KA Flow Chart for the Processing of Non–Repeat Logic and Arithmetic Tests

AS	Accumulator Sign flip–flop; AS holds the sign of the operand stored in the Accumulator; AS high indicates negative sign.
DRA	Done Repeat Arithmetic operations; DRA is sent to KM at the termination of repeat operations for L2, L3, A0–7 commands.
Fix SM	SM is adjusted according to the rules described in Table 13.1–2.
Group 2e	Term used to gate commands N2–7, A2–7, T0–7, L2, L3, S2 and S3 through KC.
Group 2f	Term used to gate commands A0, A1, T0 and T1 through KC.
KR	Opcode DONE signal to KC.
LE	Large Exponent; LE is equivalent to state QSP of the QS sequencer.
MNA	Memory Non–existent Address signal; MNA is sent by QM on detection of a non–legal address during access.
QSF ∧ DQS	Decoding indicating that the (N) = 0 and QS has taken an early exit path. Under these conditions, the path ES(0)EA is enabled to leave the exponent properly positioned in EA.
S2, S3	Logic test commands ISN and IUO which are processed by KA.
Set JNC	Request interrupt due to use of non–existent address.
SM	SuM flip–flop; SM is set to direct QS to perform a summing operation,reset for differencing.
SZ	S register Zero signal; SZ is high when register S contains zeros.
T0–7	Arithmetic test commands.
TKC	Tilt KC; TKC ⇒ Set CD8, Set KR1, Reset LP, thus enabling interrupts and disabling logic accesses. This exit is taken when KA is halted by the occurrence of a non–legal memory address.
WS	Working Sign flip–flop; WS holds the sign of the operand currently held in OA.
ZE	Zero Exponent signal; ZE comes high in KA when the EA exponent register contains zeros.

FIGURE 13.7–6 The KA Flow Chart for Processing Logic and Arithmetic Tests Non–Repeat

KA

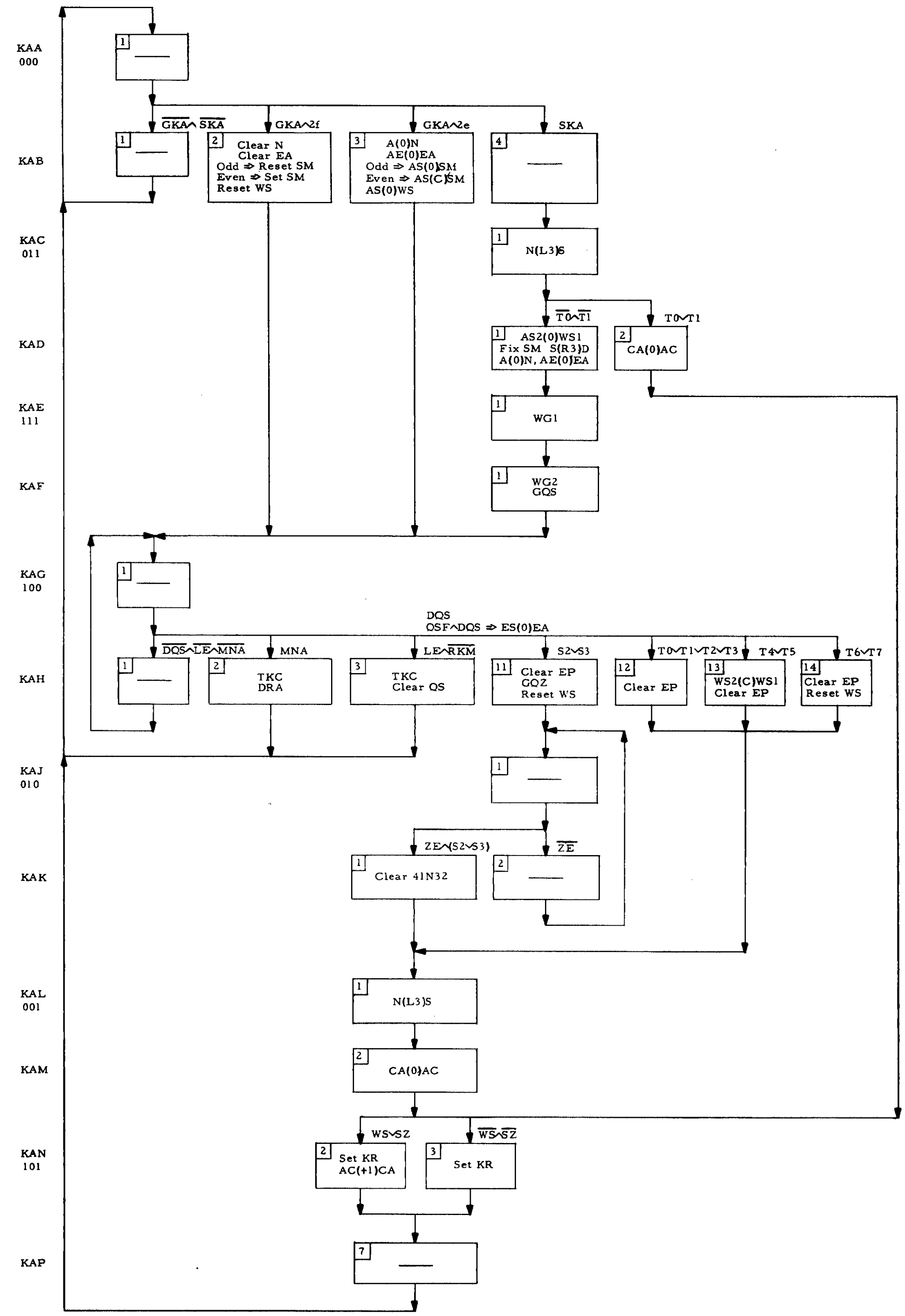


FIGURE 13.7–7 Algorithm for the Processing of Repeat Logic and Arithmetic Tests Handled by KA

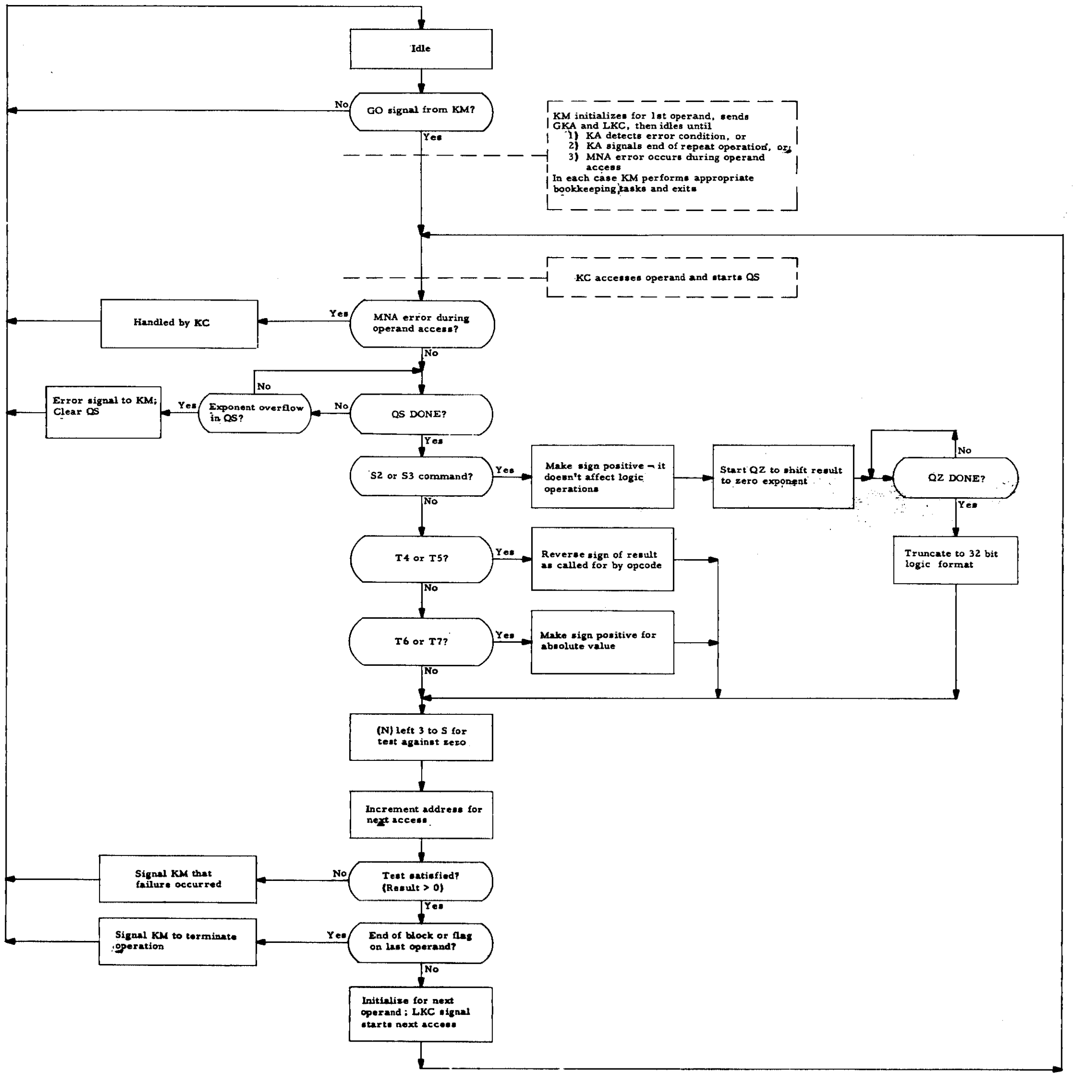

Idle
GO signal from KM?
No
Yes
KM initializes for 1st operand, sends GKA and LKC, then idles until
1) KA detects error condition, or
2) KA signals end of repeat operation, or
3) MNA error occurs during operand access
In each case KM performs appropriate bookkeeping tasks and exits
KC accesses operand and starts QS
MNA error during operand access?
Yes
Handled by KC
No
QS DONE?
No
Exponent overflow in QS?
Yes
Error signal to KM; Clear QS
No
Yes
S2 or S3 command?
Yes
Make sign positive — it doesn't affect logic operations
Start QZ to shift result to zero exponent
QZ DONE?
No
Yes
Truncate to 32 bit logic format
No
T4 or T5?
Yes
Reverse sign of result as called for by opcode
No
T6 or T7?
Yes
Make sign positive for absolute value
No
(N) left 3 to S for test against zero
Increment address for next access
Test satisfied? (Result > 0)
No
Signal KM that failure occurred
Yes
End of block or flag on last operand?
Yes
Signal KM to terminate operation
No
Initialize for next operand; LKC signal starts next access

TABLE 13.7–5 Terms Used on the KA Flow Chart for the Processing of Repeat Logic and Arithmetic Tests

AS	Accumulator Sign flip–flop; AS holds the sign of the operand stored in the Accumulator; AS high indicates negative sign.
BRA	BRAnch flip–flop; BRA is set on termination of a test command if•the termination is due to processing of the last operand in the block or the occurrence of an enabled logic flag (S2 and S3) or data flag (T0–7). The program will continue with the execution of the next command. BRA is reset if termination is due to test failure; the program will continue with execution of the next command plus 1.
DQS	DONE signal from the QS sequencer.
DRA	Done Repeat Arithmetic operation; DRA is sent to KM at the termination of repeat operations for L2, L3, A0–7 commands.
DRL	Done Repeat operation signal sent to KM when termination of a repeat test command is due to enabled flag or end of block.
DRT	Done Repeat Test operation signal sent to KM when termination of a repeat test command is due to test failure.
EVEN	Even–numbered opcodes.
Group 2a	Term used to gate commands L2, L3, S2, S3 through KC.
Group 2e	Term used to gate commands N2–7, A2–7, T0–7, L2, L3, S3 and S2 through KC.
Group 2f	Term used to gate commands A0, A1, T0 and T1 through KC.
GKA	GO signal to KA sequencer.
LE	Large Exponent; LE is equivalent to state QSP of the QS sequencer.
MNA	Memory Non–existent Address signal; MNA is sent by QM on detection of a non–legal address during access.
ODD	Odd–numbered opcodes.
RTJ	Repeat Termination signal; RTJ is high when the end of the block is reached or an enabled flag (logic for L2, L3, S2, and S3 commands, data for A or T commands) is detected.
S2, S3	Logic test commands ISN and IUO which are processed by KA.
SM	SuM flip–flop; SM is set to direct QS to perform a summing operation, reset for differencing.
SZ	S register Zero signal; SZ is high when register S contains zeros.
T0–7	Arithmetic test commands.
TKC	Tilt KC; TKC => Set CD8, Set KR1, Reset LP, thus enabling interrupts and disabling logic accesses. This exit is taken when KA is halted by the occurrence of a non–legal memory address.
WS	Working Sign flip–flop; WS holds the sign of the operand currently held in OA.
ZE	Zero Exponent signal; ZE comes high in KA when the EA exponent register contains zeros.

FIGURE 13.7–8 The KA Flow Chart for the Processing of Repeat Logic and Arithmetic Tests Handled by KA

KA

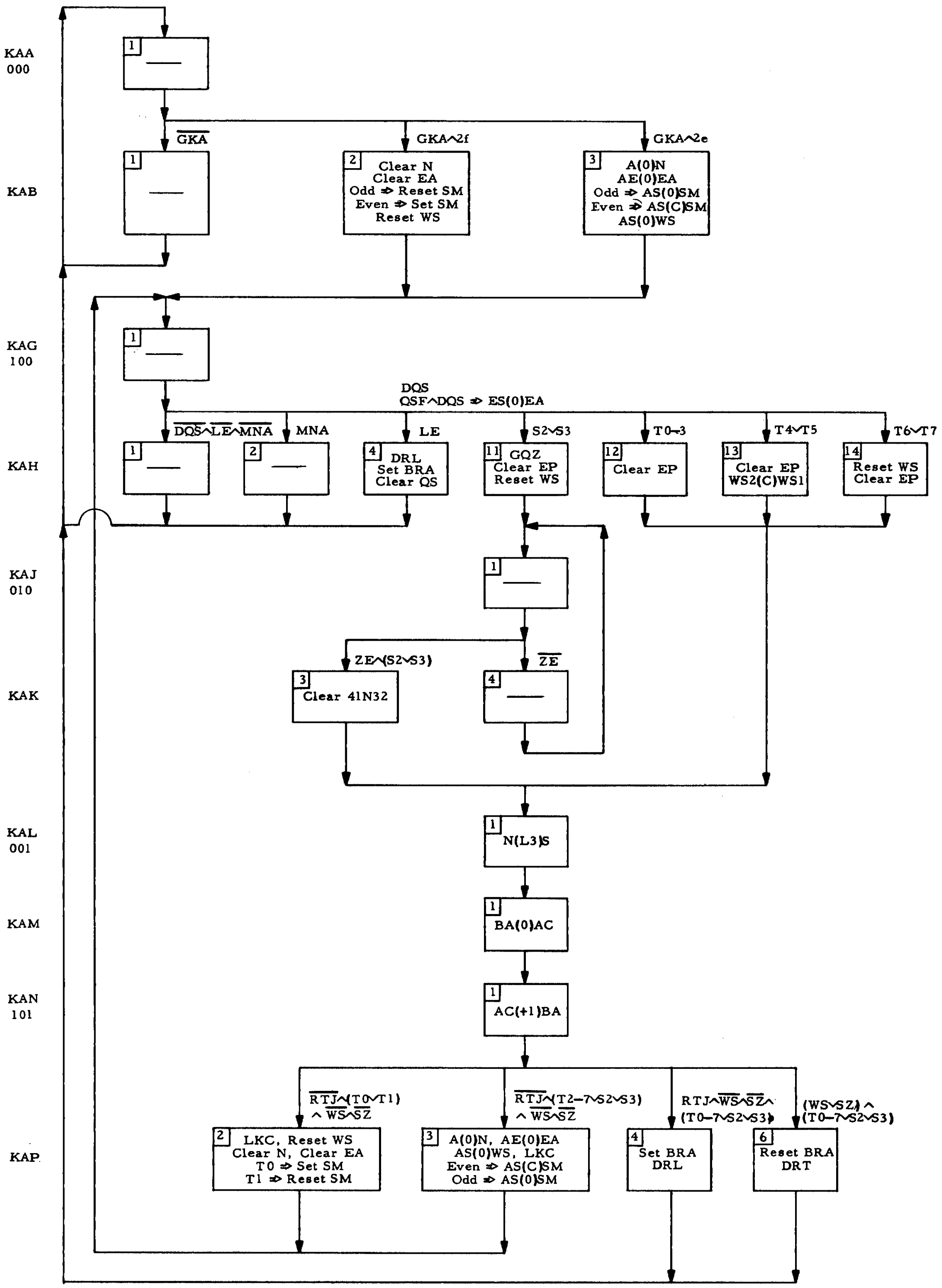
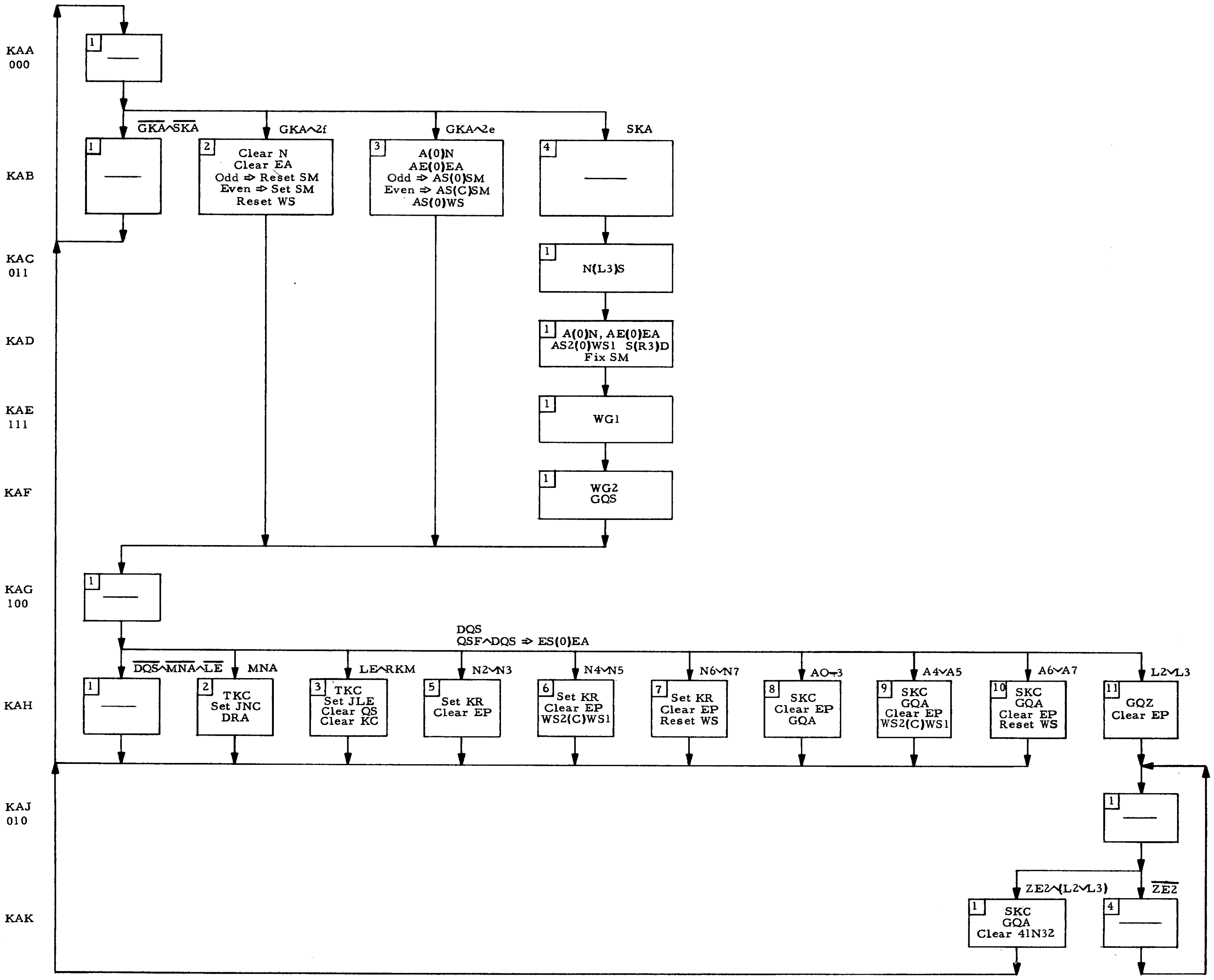


TABLE 13.7–6 Terms Used on the KA Flow Chart

A0–7	Arithmetic commands.
AS	Accumulator Sign flip–flop; AS holds the sign of the operand stored in the Accumulator; AS high indicates negative sign.
BRA	BRAnch flip–flop; BRA is set on termination of a test command if the termination is due to processing of the last operand in the block or the occurrence of an enabled logic flag (S2 and S3) or data flag (T0–7). The program will continue with the execution of the next command. BRA is reset if termination is due to test failure; the program will continue with execution of the next command plus 1.
DQS	DONE signal from the QS sequencer.
DRA	Done Repeat Arithmetic operation; DRA is sent to KM at the termination of repeat operations for L2, L3, A0–7 commands.
DRL	Done Repeat operation signal sent to KM when termination of a repeat test command is due to enabled flag or end of block.
DRT	Done Repeat Test operation signal sent to KM when termination of a repeat test command is due to test failure.
EVEN	Even–numbered opcodes.
Fix SM	SM is adjusted according to the rules described in Table 13.1–2.
Group 2a	Term used to gate commands L2, L3, S2, S3 through KC.
Group 2e	Term used to gate commands N2–7, A2–7, T0–7, L2, L3, S2 and S3 through KC.
Group 2f	Term used to gate commands A0, A1, T0 and T1 through KC.
GKA	GO signal to KA sequencer.
KR	Opcode DONE signal to KC.
L2, L3	Logic commands ADL and SUL which are processed by KA.
LE	Large Exponent; LE is equivalent to state QSP of the QS sequencer.
LKC	Loop KC; when high, LKC directs KC to access the next operand in repeat mode.
MNA	Memory Non–existent Address signal; MNA is sent by QM on detection of a non–legal address during access.
N2–7	Address Preparation commands.
ODD	Odd–numbered opcodes.
QSF ^ DQS	Decoding indicating that the (N) = 0 and QS has taken an early exit path. Under these conditions, the path ES(0)EA is enabled to leave the exponent properly positioned in EA.
RQM	READY OM sequencer.
RTJ	Repeat Termination signal; RTJ is sent when the end of the block is reached or an enabled flag (logic for L2, L3, S2, S3 commands, data for A and T commands) is detected.
S2, S3	Logic test commands ISN and IUO which are processed by KA.
Set JNC	Request interrupt due to use of non–existent address.
SKA	Start KA signal from KC; this signal is non–existent; it is equivalent to RML and indicates that the current command is non–repeat and has been written in mode 0 or 1.
SKC	Start KC; early Go to Master Control which places KC in the KCC state.
SM	SuM flip–flop; SM is set to direct QS to perform a summing operation, reset for differencing.
SZ	S register Zero signal; SZ is high when register S contains zeros.
T0–7	Arithmetic test commands.
TKC	Tilt KC; TKC ⇒ Set CD8, Set KR1, Reset LP, thus enabling interrupts and disabling logic accesses. This exit is taken when KA is halted by the occurrence of a non–legal memory address.
WG1	Wait Gate signal; WG1 is sent to the idle state of QS to enable the N(L3)Ṡ path.
WG2	Wait Gate signal; WG2 is sent to QS along with GQS to start the QS sequencer. WG2 is not the echo of WG1.
WS	Working Sign flip–flop; WS holds the sign of the operand currently held in OA.
ZE	Zero Exponent signal; ZE comes high in KA when the EA exponent register contains zeros.

FIGURE 13.7–9 The KA Flow Chart

KA



SECTION 13.8 – INTRODUCTION TO MULTIPLY AND DIVIDE

The KD, QP and QQ sequencers execute the multiply and divide opcodes. The two G–20 divide commands, divide and reverse divide, are processed by KD and QQ. (These are handled as the same command after the set up of the numerator and denominator and formation of the initial quotient exponent since, for reverse divide, the roles of the operands are simply reversed.) The single multiply command is handled by KD and QP. The KD sequencer control of multiply is discussed in Section 13.9, the product formation of QP, in Section 13.10. Similarly, KD control of division is described in Section 13.11, the quotient formation of QQ, in Section 13.12. (This results in a split discussion of KD: the entire KD flow chart is included at the end of Section 13.11 for the sake of completeness.) The G–20 algorithms for the processing of multiply and divide commands are not discussed here. This section is included for purposes of orientation. The similarities between the two operations are described with particular emphasis on use of the Adder and exponent manipulation.

Considering the general case of multiplication and division in binary arithmetic rather than the handling of these operations by the G–20, the following statements can be made. For each multiply or divide operation the same set of actions is repeated several times, this because multiplication is actually a sequence addition operation, while divide is a sequence subtract operation. During multiply, the two operands involved in the addition are the multiplicand and the partial product. The result of each addition is the new partial product. (For the first add cycle, the partial product value is zero.) The final value is the last partial product, called simply the product.

As an example of what happens during binary multiplication, consider the case where 2 is multiplied by 2:

multiplicand = 2 = 010
multiplier = 2 = 010

The bits in the multiplier determine when add cycles are to take place. Each bit in the multiplier, starting with the least significant, is examined. The least significant position, or the first order, in the example contains a 0 bit; therefore, no add cycle takes place. The second order contains a 1, calling for an addition. The Adder inputs in this case will be a zero partial product and the multiplicand which will be shifted 1 bit to the left to position it at the same order as the multiplier bit under consideration. The third order multiplier bit is again a zero. Thus, only one add cycle is necessary, with the inputs being:

multiplicand 100 (shifted 1 bit to the left)
partial product 000
Adder output 100 = 4

In divide, the two operands involved in the subtraction are the denominator and the numerator (or remainder). Each remainder is formed by subtracting the denominator from the last remainder whenever r > d. The remainder is compared to the denominator starting with the most significant bit. For each such subtraction, the result of the operation (quotient) receives a 1 bit in the appropriate order; when no subtraction takes place, the quotient receives a 0 bit. (The quotient is formed starting with the most significant bit.) Thus, if 4 is divided by 2, the operands are as follows:

remainder (numerator) = 4 = 100
denominator = 2 = 010

At the most significant order, r is 1 while d is 10, so the quotient bit is zero. At the second most significant order, r is 10 and d is 10 so a subtraction takes place:

$$\begin{array}{r} 10 \\ \underline{-10} \\ 00 \end{array}$$

and the quotient receives a 1. The quotient is now 01. At the third order, the remainder is now 00 while the denominator is 10 so that no subtraction takes place and the quotient receives a 0. Therefore, the final quotient is 010 or 2. Thus, in multiply, the least significant multiplier bits are examined first; in divide, the most significant remainder bits are examined first.

The relationships between the operands involved in these operations and the way in which the G—20 Adder inputs are set up is shown in Table 13.8—1.

In both cases it is necessary to shift one of the Adder inputs between each add cycle in order to position the operands at the correct order. It is possible to position these operands in more than one way and get the correct result if care is taken to keep the exponent value correct. For example if, in multiply, the multiplicand is positioned to be added to the correct order of the partial product at each addition by means of shifting the multiplicand to the left, no change is required in the exponent value. If, instead, the partial product is shifted to the right, the product exponent is incremented in step with the shifts. In divide, if the denominator is shifted right prior to each subtraction, no exponent change is required. However, if the remainder is shifted to the left, the quotient exponent is decremented in step with the shifts. As will be explained in the following sections, the G—20 logic handles both operations in the non—standard manner. Thus, for multiplication, the partial product is shifted right and the exponent incremented; for

TABLE 13.8–1 Use of the Adder in Multiply and Divide Operations

Operation:	MULTIPLY	DIVIDE
Role of assembled operand:	multiplier	denominator (divide) numerator (reverse divide)
Role of operand from Accumulator:	multiplicand	numerator (divide) denominator (reverse divide)
Adder inputs:	multiplicand and partial product	remainder (numerator on first subtraction) and denominator
Add cycle started on basis of:	examination of multiplier bits	relative sizes of remainder and denominator
Output from Adder:	new partial product: on last cycle, final product	new remainder
Final result of operation:	final product	quotient formed during operation; quotient receives a 1 bit at each order where a sub–traction occurs, 0 bits otherwise.

division, the remainder is shifted left and the exponent decremented.

An additional action that is repeated in each operation involves the manipulations performed on the operand that is not an input to the Adder. During multiplication, this is the multiplier, the operand whose bit configuration determines the orders at which add cycles take place. For divide, this is the quotient which is formed by the insertion of 1 bits at each order for which a subtraction occurs. It would be possible, though difficult, to design the logic to handle these values without shifting them. However, it is far more satisfactory to shift the values so that the examination of multiplier bits or insertion of quotient bits always occurs at the same place. Thus, the multiplier is shifted to the right as the operation progresses and the least significant bits examined. This shifting is in step with that of the partial product and in fact, governs the shifts of the partial product. Conversely, the quotient is shifted to the left with the new bits being inserted into the least significant bit position. Multiplication continues until the multiplier is shifted out of existence; division continues until a quotient of 14 octal digits has been formed.

From this brief analysis it can be seen that in multiply and divide operations there is one operand that requires no shifting, two that do require shifting. Since these shifts occur between gatings of the partial product or remainder from the Adder, the choice of shift paths has been based on the Adder input requirements. The Adder input registers are N and D and, therefore, the quantities that must be summed or differenced, the partial product and multiplicand or the remainder and denominator, are set up in these registers. The Adder output (the current partial product for multiply or the current remainder for divide) is gated to register S. Since the multiply algorithm calls for right shifts of the partial product, and right shift paths are available between S and D, this dictates storage of the partial product in D, thereby delegating the multiplicand to N. Similarly, N becomes the storage register for the remainder in divide since the remainder is shifted to the left and left shift paths are available between S and N. The A and M registers are used to shift the multiplier during multiply and to form the quotient during divide.

Thus, it is clear that shifting is a basic factor in these operations and that each shift affects the value of the final exponent. A preliminary exponent is formed by the QA sequencer prior to setting up the operation. Following that, each time a shift is performed on a quantity involved in the operation, the exponent is counted up or down accordingly. This counting may be octal or binary depending upon whether the shift is 1 or 3 bits. Since shifts occur in both directions, it is necessary to count the exponents up and down.

Octal shifts with concurrent exponent increments or decrements of 1 occur frequently in opcode processing. This facility is provided by the octal exponent counter which consists of registers ES and EA. It counts up or down via the paths ES(+1)EA and ES(—1)EA. (The same paths are correct for negative exponents since these values are stored in 1's complement form.) In multiply and divide, this counter is activated by the signal CHE. It is set up to accept CHE on C2's but decoding takes place on C1's, making it necessary to employ a change count delay flip—flop (CED) to hold off the CHE signal for 1/2 clock. Thus, a change in the octal count is implemented by the setting of CED which in turn sets CHE causing the relevant increment or decrement.

The change octal exponent circuitry adjusts the exponent for the octal shifts in multiply and divide. However, these operations also call for frequent binary shifts which necessitate exponent increments and decrements of 1/3. The two registers used for this purpose are MM and MT, referred to jointly as the modulo—three counter. For 1 bit shifts to the right, the signal MT(+1)MM causes the counter to be incremented by 1/3; for left shifts of 1 bit, the signal is MT(—1)MM. At the next clock time, the contents of MM are copied back to MT via the MM(0)MT path. When incrementing, the counting sequence is 00, 01, 10, 00: when decrementing, 00, 10, 01, 00.

The modulo—three counter affects the octal exponent in EA when CED is set. Since the exponents count up and down in both operations, it is necessary to make the action taken on receipt of CED (i.e., the decision to count up or down) a function of the current opcode and sequencer state. If these considerations indicate an exponent increment, the CEP (Change Exponent Positive) signal goes high; if a decrement is indicated, CEN (Change Exponent Negative) goes high. Thus, if the counter contains 10 and an MT(+1)MM signal is received, CED will be decoded along with the current sequencer state to set CEP and the ES(+1)EA path will be enabled.

Both multiply and divide require checking of exponent ranges. The normal range of exponents, from –63 to +63, can be held in a 6–bit register; detection of over and underflows is effected by a seventh bit, and of over–overflows and under–underflows by an eighth bit. These bits also permit proper counting outside of the normal range so that, when checking shows that such a condition exists, an attempt to retrieve the value through shifting can be made. Two things should be pointed out: one, if an overflow is detected, an interrupt will be requested. This is true of all G–20 operations. Two, if an under–underflow condition exists, it will be impossible to retrieve the value through shifting right since the value will become zero before the exponent reaches the normal range. Thus, detection of under–underflows results in a normal zero replacing the final value as the result. For an underflow condition, if the mantissa is shifted to zero before the exponent reaches the normal range, a normal zero is the final result. Provisions for counting outside the normal range make it possible to hold off exponent range checking until the final stage in multiply/divide operations.

Thus, a normal exponent of +63 appears as follows in the EA register:

$$\begin{array}{r|ccccccccc|} & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ \cline{2-10} (EA) = & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ \cline{2-10} \end{array}$$

The ninth bit, EA8, is set to 0 for positive.

Table 13.8–2 demonstrates the use of the most significant bits in the EA register.

TABLE 13.8–2 Exponent Ranges and Associated Signals

Signal	Range	EA8	ES7	ES6	Condition Indicated
EVV	+128 to +191	0	1	0	Exponent oVer–oVerflow
EVF	+ 64 to +127	0	0	1	Exponent oVerFlow
EXN	0 to + 63	X	0	0	EXponent Normal
	– 63 to – 1	X	1	1	
EUF	–127 to – 64	1	1	0	Exponent UnderFlow
EUU	–191 to –128	1	0	1	Exponent Under–Underflow

X's appear for EA8 in the EXN range because EA8 is fixed before any counting takes place and counting could cause the counter to change to the upper half of the EXN range from the lower or vice versa, thus reversing the state of EA8.

Having covered the areas common to the multiply and divide operations, it is possible to proceed to a description of the basic multiply algorithm and its implementation without unnecessary redundancy.

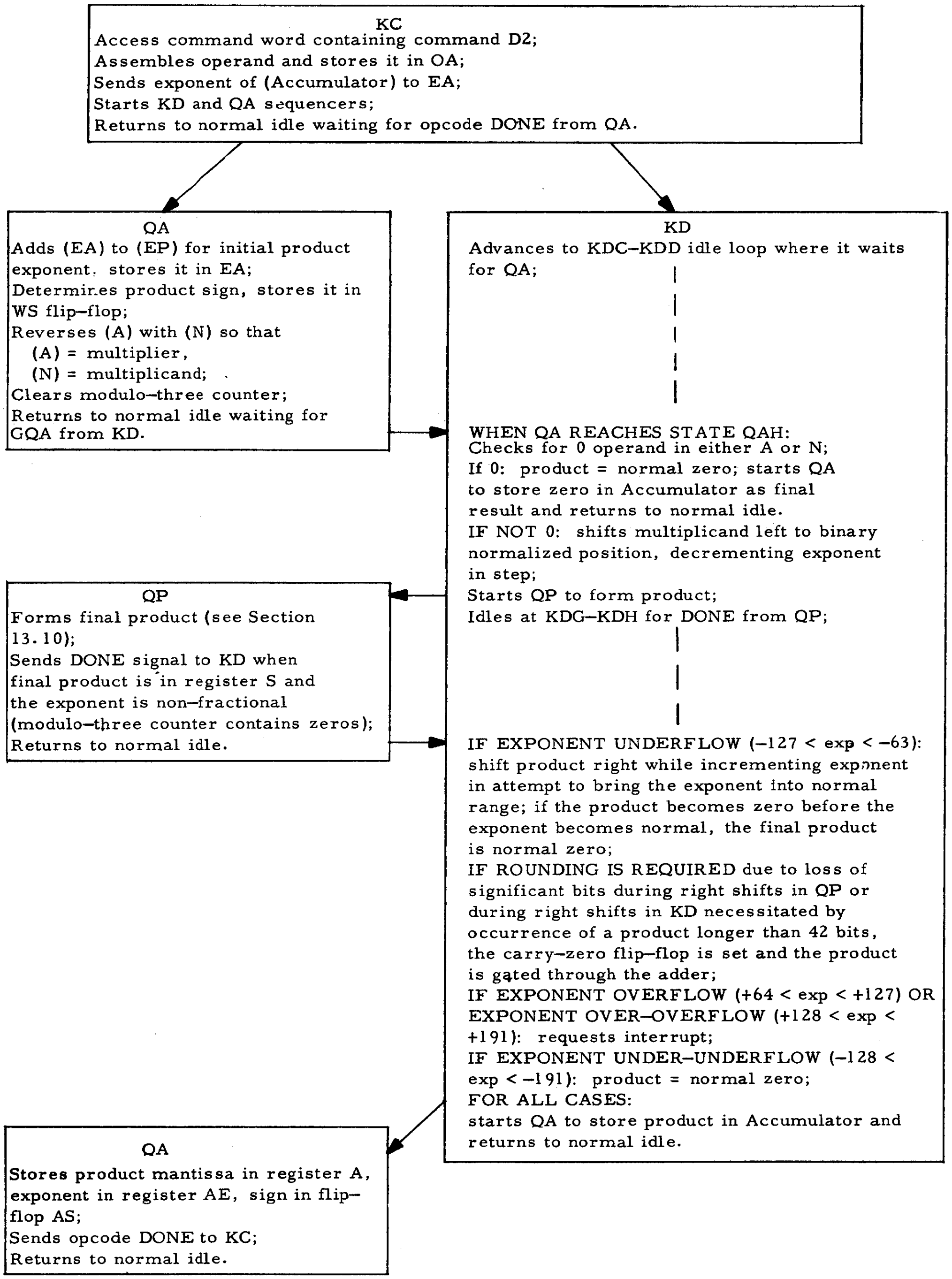
SECTION 13.9 – KD SEQUENCER CONTROL OF MULTIPLY OPERATIONS

Product formation during multiplication is handled by the QP sequencer. Prior to the startup of QP, the QA and KD sequencers set up the operation. The interaction between these sequencers is shown in Figure 13.9–1. Note that QA sums the exponents of the operands for an initial product exponent value and determines the sign of the product. This sign is positive if both operands (the multiplier and the multiplicand) have the same sign whereas, if their signs differ, the product sign is negative. The final product sign is determined and stored in the WS flip–flop by means of complementing the sign in WS (the sign of the multiplier) if the sign in flip–flop AS (the sign of the multiplicand) is negative. Other sign manipulations occur during the operation of QP, but have no bearing on the sign of the final product. The state of WS does not change once it is established by QA.

When QA has transferred the multiplicand to register N, KD checks both operands against zero. If either is a zero, KD starts QA and exits, leaving a zero in register N to be stored in the Accumulator by QA as the final result. Otherwise, KD proceeds by shifting the multiplicand left until it is binary normalized, i.e., the most significant bit in the register is a 1. The product exponent is decremented in step with these left shifts. (KD shifts octally, decrementing the exponent by 1 for each shift, until octal normalization occurs, then shifts binarily with exponent decrements of 1/3.) The multiplicand is positioned in this manner to allow for retention of maximum significance in the final product. The first partial product is formed by the addition of the multiplicand to the initial partial product value of zero; thus, the binary normalization of the multiplicand means that the first partial product will also be binary

FIGURE 13.9–1 General Algorithm for Multiply Operations

D2 (MPY) calls for the multiplication of the quantity stored in the Accumulator by the operand specified in the command word. Thus, (OA) becomes the multiplier, (Accumulator), the multiplicand. Processing is as follows:



KC
Access command word containing command D2;
Assembles operand and stores it in OA;
Sends exponent of (Accumulator) to EA;
Starts KD and QA sequencers;
Returns to normal idle waiting for opcode DONE from QA.

QA
Adds (EA) to (EP) for initial product exponent, stores it in EA;
Determines product sign, stores it in WS flip–flop;
Reverses (A) with (N) so that
(A) = multiplier,
(N) = multiplicand;
Clears modulo–three counter;
Returns to normal idle waiting for GQA from KD.

KD
Advances to KDC–KDD idle loop where it waits for QA;

WHEN QA REACHES STATE QAH:
Checks for 0 operand in either A or N;
If 0: product = normal zero; starts QA to store zero in Accumulator as final result and returns to normal idle.
IF NOT 0: shifts multiplicand left to binary normalized position, decrementing exponent in step;
Starts QP to form product;
Idles at KDG–KDH for DONE from QP;

QP
Forms final product (see Section 13.10);
Sends DONE signal to KD when final product is in register S and the exponent is non–fractional (modulo–three counter contains zeros);
Returns to normal idle.

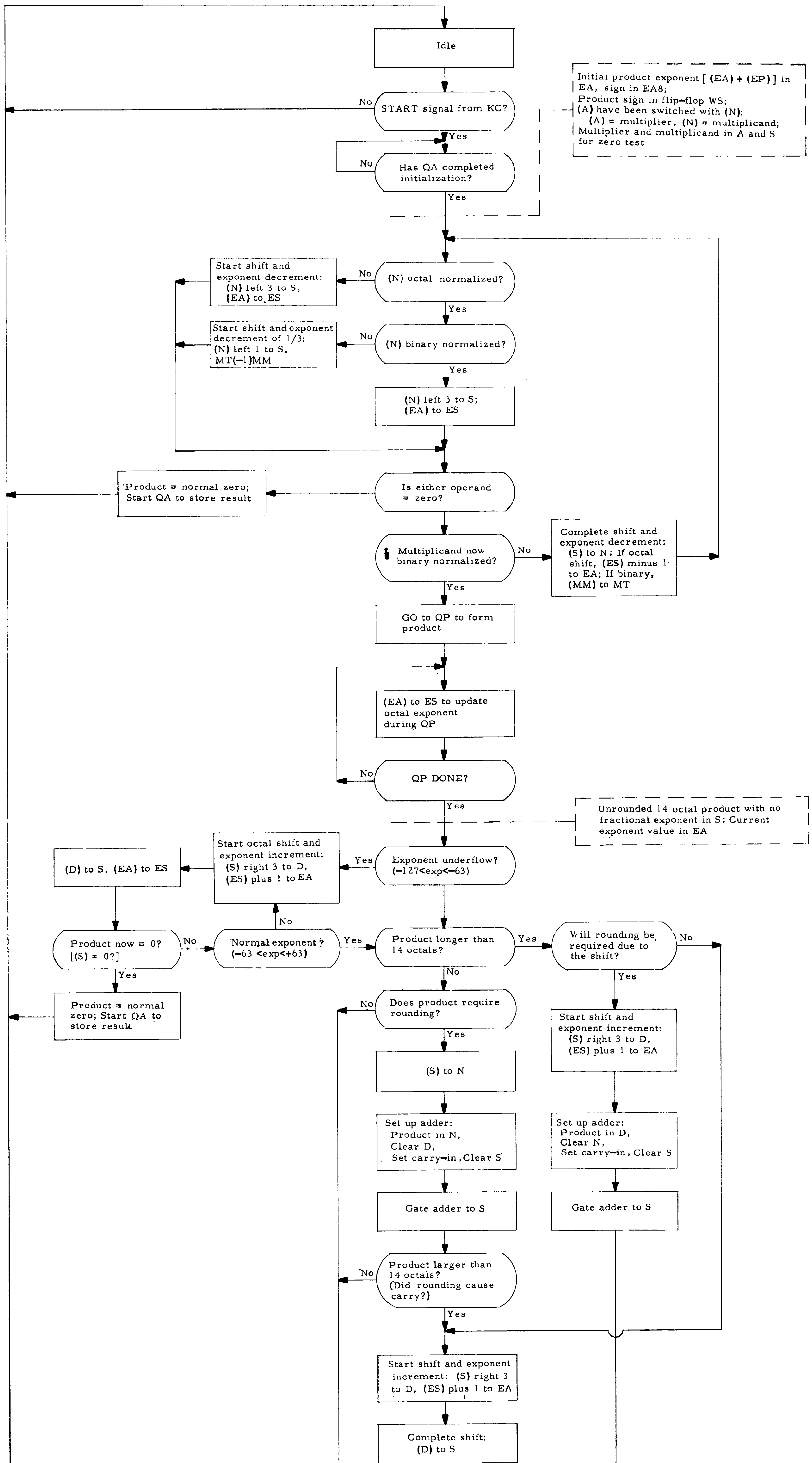
IF EXPONENT UNDERFLOW (–127 < exp < –63): shift product right while incrementing exponent in attempt to bring the exponent into normal range; if the product becomes zero before the exponent becomes normal, the final product is normal zero;
IF ROUNDING IS REQUIRED due to loss of significant bits during right shifts in QP or during right shifts in KD necessitated by occurrence of a product longer than 42 bits, the carry–zero flip–flop is set and the product is gated through the adder;
IF EXPONENT OVERFLOW (+64 < exp < +127) OR EXPONENT OVER–OVERFLOW (+128 < exp < +191): requests interrupt;
IF EXPONENT UNDER–UNDERFLOW (–128 < exp < –191): product = normal zero;
FOR ALL CASES:
starts QA to store product in Accumulator and returns to normal idle.

QA
Stores product mantissa in register A, exponent in register AE, sign in flip–flop AS;
Sends opcode DONE to KC;
Returns to normal idle.

normalized. As product formation proceeds, this partial product is shifted to the right. Thus, the initial left justification of the value results in a minimum loss of significance.

KD starts QP to form the product, then idles until the DONE QP signal is received. At this time the unrounded, 14-octal product is in register S with a non-fractional exponent (modulo-three counter contains zeros) stored in register EA. KD performs exponent range checking and rounds the product if significant bits have been lost due to right shifts. Finally, KD starts QA to store the result and returns to idle.

FIGURE 13.9–2 Algorithm for KD Control of Multiply Operations


Idle
START signal from KC?
No
Yes
Has QA completed initialization?
No
Yes
Initial product exponent [(EA) + (EP)] in EA, sign in EA8;
Product sign in flip-flop WS;
(A) have been switched with (N):
(A) = multiplier, (N) = multiplicand;
Multiplier and multiplicand in A and S for zero test
(N) octal normalized?
No
Start shift and exponent decrement: (N) left 3 to S, (EA) to ES
Yes
(N) binary normalized?
No
Start shift and exponent decrement of 1/3: (N) left 1 to S, MT(–1)MM
Yes
(N) left 3 to S; (EA) to ES
Is either operand = zero?
Product = normal zero; Start QA to store result
Multiplicand now binary normalized?
No
Complete shift and exponent decrement: (S) to N; If octal shift, (ES) minus 1 to EA; If binary, (MM) to MT
Yes
GO to QP to form product
(EA) to ES to update octal exponent during QP
QP DONE?
No
Yes
Unrounded 14 octal product with no fractional exponent in S; Current exponent value in EA
Exponent underflow? (–127<exp<–63)
Yes
Start octal shift and exponent increment: (S) right 3 to D, (ES) plus 1 to EA
(D) to S, (EA) to ES
Product now = 0? [(S) = 0?]
No
Normal exponent? (–63 <exp<+63)
No
Yes
Product = normal zero; Start QA to store result
Yes
Product longer than 14 octals?
Yes
Will rounding be required due to the shift?
No
Yes
Start shift and exponent increment: (S) right 3 to D, (ES) plus 1 to EA
Set up adder: Product in D, Clear N, Set carry-in, Clear S
Gate adder to S
No
Does product require rounding?
No
Yes
(S) to N
Set up adder: Product in N, Clear D, Set carry-in, Clear S
Gate adder to S
Product larger than 14 octals? (Did rounding cause carry?)
No
Yes
Start shift and exponent increment: (S) right 3 to D, (ES) plus 1 to EA
Complete shift: (D) to S

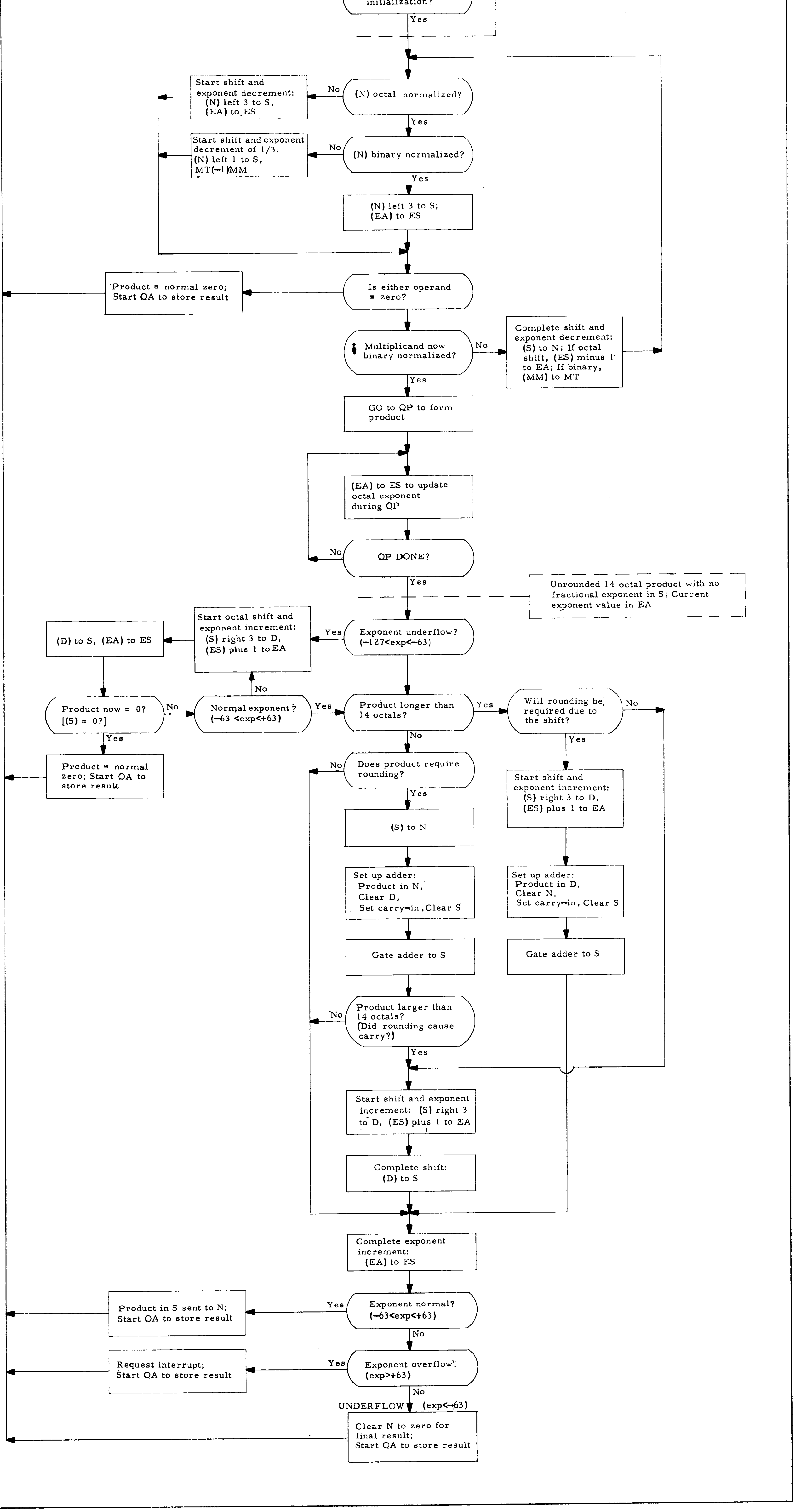
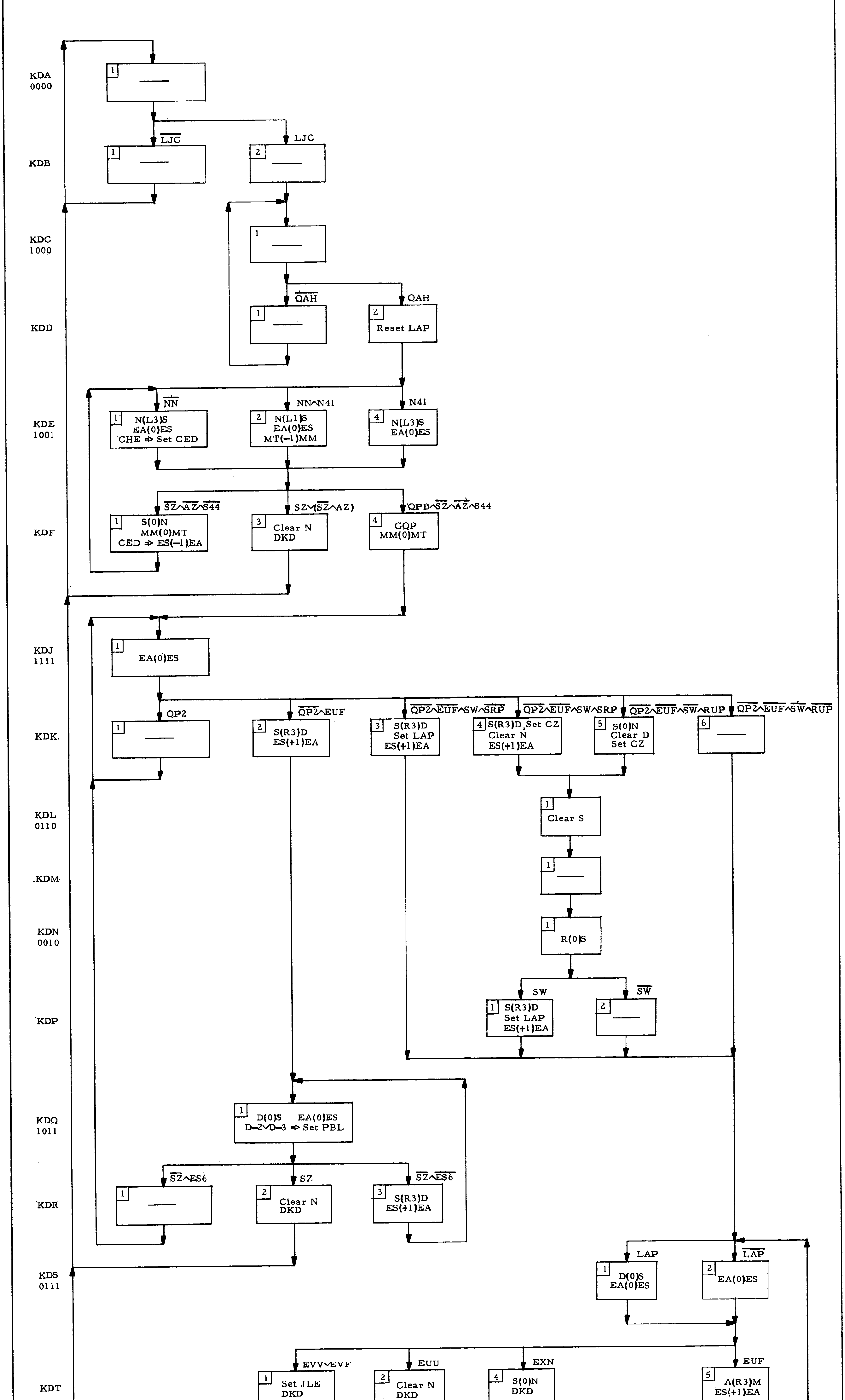

initialization?
Yes
Start shift and exponent decrement: (N) left 3 to S, (EA) to ES
No
(N) octal normalized?
Yes
Start shift and exponent decrement of 1/3: (N) left 1 to S, MT(−1)MM
No
(N) binary normalized?
Yes
(N) left 3 to S; (EA) to ES
Product = normal zero; Start QA to store result
Is either operand = zero?
Multiplicand now binary normalized?
No
Complete shift and exponent decrement: (S) to N; If octal shift, (ES) minus 1 to EA; If binary, (MM) to MT
Yes
GO to QP to form product
(EA) to ES to update octal exponent during QP
No
QP DONE?
Yes
Unrounded 14 octal product with no fractional exponent in S; Current exponent value in EA
(D) to S, (EA) to ES
Start octal shift and exponent increment: (S) right 3 to D, (ES) plus 1 to EA
Yes
Exponent underflow? (−127<exp<−63)
No
Product now = 0? [(S) = 0?]
No
Normal exponent ? (−63 <exp<+63)
Yes
Product longer than 14 octals?
Yes
Will rounding be required due to the shift?
No
Yes
Product = normal zero; Start QA to store result
No
No
Does product require rounding?
Yes
Yes
Start shift and exponent increment: (S) right 3 to D, (ES) plus 1 to EA
(S) to N
Set up adder: Product in N, Clear D, Set carry-in, Clear S
Set up adder: Product in D, Clear N, Set carry-in, Clear S
Gate adder to S
Gate adder to S
No
Product larger than 14 octals? (Did rounding cause carry?)
Yes
Start shift and exponent increment: (S) right 3 to D, (ES) plus 1 to EA
Complete shift: (D) to S
Complete exponent increment: (EA) to ES
Product in S sent to N; Start QA to store result
Yes
Exponent normal? (−63<exp<+63)
No
Request interrupt; Start QA to store result
Yes
Exponent overflow? (exp>+63)
No
UNDERFLOW (exp<−63)
Clear N to zero for final result; Start QA to store result

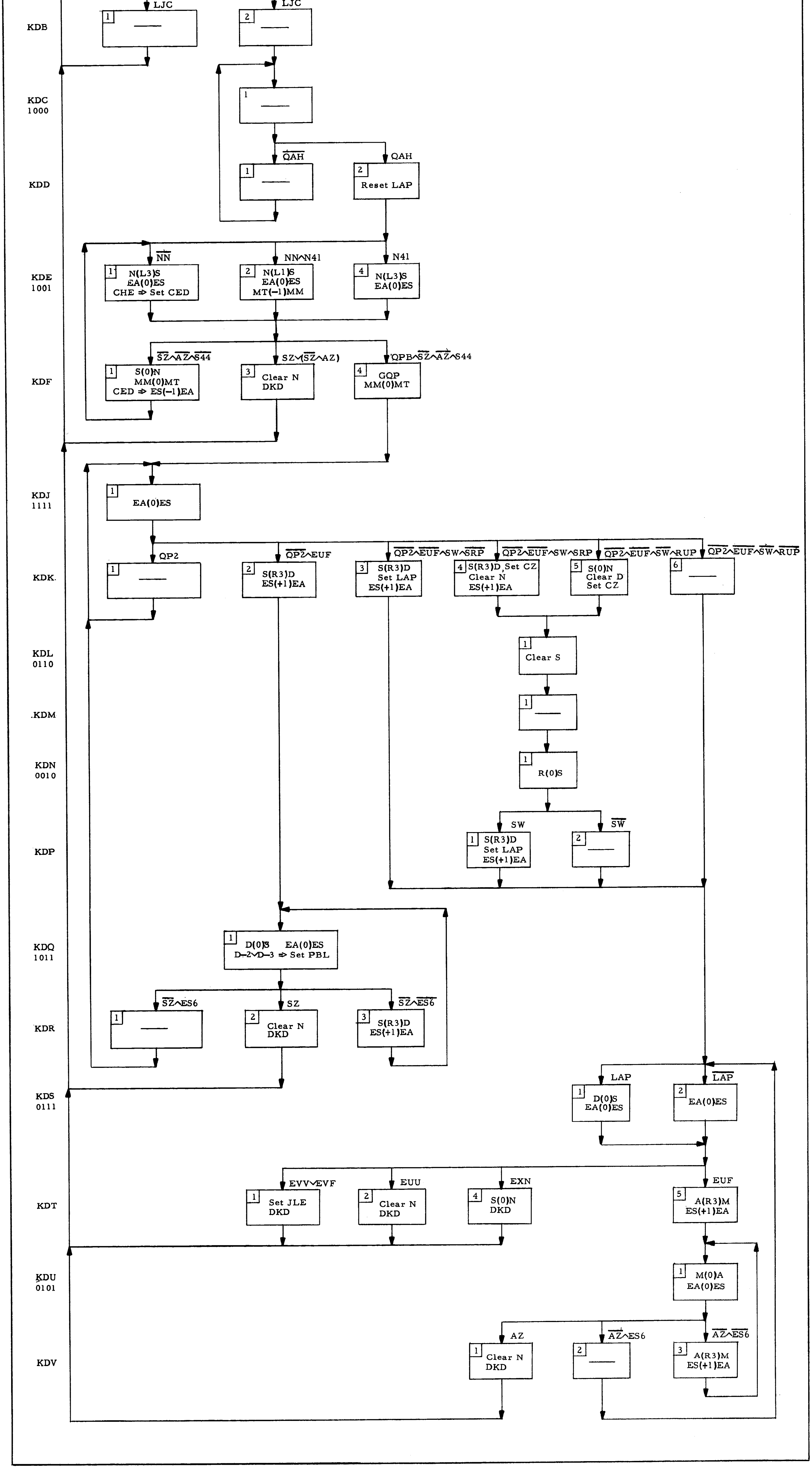
TABLE 13.9–1 Terms Used on the KD Flow Chart for Control Multiply

AZ	A register Zero; AZ is high when the contents of register A are zero.
CD10	Command Decoding bit used to distinguish between multiply and divide; CD10 is high for multiply.
CHE ⇒ Set CED	Path enabled to effect octal exponent decrement during octal normalization of the multiplicand. (CHE = CHange octal Exponent signal; CED = Change Exponent Delay signal that will, in this case, enable CEN to go high, which will be used for enabling of the ES(−1)EA path.)
CZ	Carry Zero flip–flop; CZ is set when a carry-in to the zero order of the Adder is desired; in KD it is set prior to roundup.
DKD	DONE KD sequencer; this signal is non–existent; it is equivalent to the following: GQA, reset CZ, clear KD.
ES6	Bit 6 in register ES; ES6 is low when an exponent underflow occurs, high otherwise.
EUF	Exponent UnderFlow signal; EUF is high when the exponent held in register ES is in the −127 to −64 range.
EUU	Exponent Under–Underflow signal; EUU is high when the exponent held in register ES is in the −191 to −128 range.
EVF	Exponent oVerFlow signal; EVF is high when the exponent held in register ES is in the +64 to +127 range.
EVV	Exponent oVer–oVerflow signal; EVV is high when the exponent held in register ES is in the +128 to +191 range.
EXN	EXponent Normal signal; EXN is high when the exponent held in register ES is in the −63 to +63 range.
GQP	GO signal to the QP sequencer.
JLE	Interrupt request set if exponent overflow has occurred.
LAP	Long Accumulator Put signal; LAP is set when the product held in the S register is longer than 42 bits, but rounding is not required; LAP gates the last step of the right–3–bits shift.
LJC	Signal sent by KC to start the KD and QA sequencers; LJC is equivalent to GO KD.
N41	Bit 41 in register N; N41 is high when there is a 1 in this bit position indicating binary normalization of the multiplicand.
NN	(N) octal Normalized signal; NN is high when significant bits occur in the most significant octal position in register N.
QAH	State of the QA sequencer; when QA has advanced to QAH, KD can continue with its operations.
QPB	Decoding for clock 2 of idle state of QP sequencer; QPB high indicates that QP is not in operation.
QP2	Decoding inverter for the QP sequencer; all states in QP except idle (QPA–QPB) have as part of their decoding QP2; thus, $\overline{\text{QP2}}$ is used to signal DONE QP sequencer.
R(0)S	Path enabled to gate product from Adder when rounding has occurred.
RUP	RoundUp Product; RUP is high when no shift is called for but rounding is necessary due to the loss of significant bits (PBL high) during product formation; thus, RUP = $\text{S}{-1} \wedge (\text{PBL} \vee \overline{\text{S0}})$.
S44	Bit 44 in register S; when S44 is high, the multiplicand is binary normalized; (this is equivalent to the N41 signal since a left 3 shift occurred between N and S).
SRP	Shift, Roundup Product; SRP is high when rounding is called for in conjunction with a 3–bit–right shift due to SW high. Thus, SRP = $\text{S2} \wedge (\text{PBL} \vee \text{S1} \vee \text{S0} \vee \text{S}{-1} \vee \overline{\text{S3}})$.
SW	S register Wide signal; SW is high when the product is too large (greater than 42 bits); this condition is corrected by means of a right 3 shift and, if SRP is high, rounding occurs. SW = $\text{S42} \vee \text{S43} \vee \text{S44}$.
SZ	S register Zero signal; SZ is high when the contents of register S are zero.

FIGURE 13. 9–3 The KD Flow Chart for Control of Multiply Operations

KD




KDB
LJC
1 ——
LJC
2 ——
KDC 1000
1 ——
KDD
$\overline{QAH}$
1 ——
QAH
2 Reset LAP
KDE 1001
$\overline{NN}$
1 N(L3)S EA(0)ES CHE ⇒ Set CED
NN∧N41
2 N(L1)S EA(0)ES MT(−1)MM
N41
4 N(L3)S EA(0)ES
KDF
$\overline{SZ}\wedge\overline{AZ}\wedge\overline{S44}$
1 S(0)N MM(0)MT CED ⇒ ES(−1)EA
$SZ\vee(\overline{SZ}\wedge AZ)$
3 Clear N DKD
$QPB\wedge\overline{SZ}\wedge\overline{AZ}\wedge S44$
4 GQP MM(0)MT
KDJ 1111
1 EA(0)ES
KDK.
QP2
1 ——
$\overline{QP2}\wedge EUF$
2 S(R3)D ES(+1)EA
$\overline{QP2}\wedge\overline{EUF}\wedge SW\wedge\overline{SRP}$
3 S(R3)D Set LAP ES(+1)EA
$\overline{QP2}\wedge\overline{EUF}\wedge SW\wedge SRP$
4 S(R3)D, Set CZ Clear N ES(+1)EA
$\overline{QP2}\wedge\overline{EUF}\wedge\overline{SW}\wedge RUP$
5 S(0)N Clear D Set CZ
$\overline{QP2}\wedge\overline{EUF}\wedge\overline{SW}\wedge\overline{RUP}$
6 ——
KDL 0110
1 Clear S
.KDM
1 ——
KDN 0010
1 R(0)S
KDP
SW
1 S(R3)D Set LAP ES(+1)EA
$\overline{SW}$
2 ——
KDQ 1011
1 D(0)S EA(0)ES D−2∨D−3 ⇒ Set PBL
KDR
$\overline{SZ}\wedge ES6$
1 ——
SZ
2 Clear N DKD
$\overline{SZ}\wedge\overline{ES6}$
3 S(R3)D ES(+1)EA
KDS 0111
LAP
1 D(0)S EA(0)ES
$\overline{LAP}$
2 EA(0)ES
KDT
EVV∨EVF
1 Set JLE DKD
EUU
2 Clear N DKD
EXN
4 S(0)N DKD
EUF
5 A(R3)M ES(+1)EA
KDU 0101
1 M(0)A EA(0)ES
KDV
AZ
1 Clear N DKD
$\overline{AZ}\wedge ES6$
2 ——
$\overline{AZ}\wedge\overline{ES6}$
3 A(R3)M ES(+1)EA

SECTION 13.10 – PRODUCT FORMATION BY THE QP SEQUENCER

In binary multiplication, the product is formed by means of adding the multiplicand to the partial product at each order in which a 1 bit occurs in the multiplier. The simplicity of binary arithmetic speeds up the process since the multiplicand itself is always the second operand. (This, of course, is due to the fact that only 0 and 1 bits are possible in the multiplier.) The decision made at each order is between shifting and adding or shifting and not adding.

At GO QP, the current value of the product exponent is held in the octal exponent counter and the modulo–three counter. Appropriate increments are made to this exponent as right shifts occur during the operation of QP. When QP is started, the operands are positioned as described in Section 13.8, that is:

1) the multiplicand is binary normalized in register N; it remains here, unchanged, throughout the operation of QP;

2) the multiplier is in the A register; it will be shifted to the right between the A and M registers.

The partial product is formed between the S and D registers. The S register is cleared by QP for the starting partial product value of zero. For each add cycle, the partial product is left in register D while the S register is cleared to receive the Adder output.

In the G–20, the multiply algorithm is speeded up by means of examination of the 3 least significant multiplier bits simultaneously, a process known as string multiplication. Depending upon the configuration of these bits, a 1–, 2–, or 3–bit shift to the right will occur in both the partial product and the multiplier. An add cycle may or may not be part of the action taken.

To begin with the simple case, consider a multiplier whose least significant 3 bits are zeros. These call for no action other than right 3 shifts of the multiplier and the partial product along with an octal exponent increment of 1.

Three 1 bits in succession can be handled easily if advantage is taken of the relationship between a number consisting entirely of 1's and the number that is one greater than this. For example, the number 1000 is 1 greater than 111 or equal to 111 + 1, just as the number 1000000 is 1 greater than 111111. Thus, if the 3 multiplier bits being examined are 1's, or 111 * multiplicand, this can equally well be viewed as 1000 * multiplicand minus 1 * multiplicand. On the basis of this, a subtraction of 1 * multiplicand takes place at the first order of the string of 1's, followed by right shifts of the partial product and the multiplier until the order following the last 1 bit is reached. At this order the quantity 1 * multiplicand is added to the partial product.

When the subtraction takes place at the first order of the string (on entry into the string) the partial product is lower than its true value and is, indeed, negative. This negative value is remembered by means of the Partial product Sign flip–flop, PS. When the end of the string is reached and the final addition occurs, the partial product is restored to its true value and PS is reversed to indicate positive. Thus, the PS flip–flop records the sign of the partial product and, thereby, indicates entry into or exit from a string of 1's. This facilitates keeping track of the fact that a long string of 1's is being processed so that they can be handled as easily as short strings. (When the 3 bits examined are all 1's and PS indicates a negative partial product or that a 1's string is being processed, the multiplier and partial product are shifted right and the exponent is incremented by 1.)

PS is initially reset, or positive, to indicate the zero's string mode, i.e., if the first 3 bits are zeros the only action taken would be right shifts and exponent increment. QP does not exit until PS is reset; this insures that a positive partial product will always be restored for a true final value. Again, this sign value does not affect the final product sign that has been established by QA and stored in the WS flip-flop. PS is used only during the operation of QP and for the purposes mentioned above.

When use of the Adder is indicated, the operation performed is a sum or difference depending upon the current sign values. These values are handled in a manner similar to QS, but the implications are less far-reaching since they are confined to the realm of QP. The PS flip-flop holds the current partial product sign; it is combined with that of the indicated addition or subtraction to determine whether to sum or difference the operands. When the signs are similar, the operation is summing; when they differ, differencing. (The multiplicand is assumed to be positive and, therefore, does not affect the determination.) Consider the case of entry into a 1's string (mode 1) which calls for subtraction of 1 * multiplicand. Since the partial product sign has been positive and this is a subtraction, a differencing operation will be performed. The new partial product will be negative. At the end of the string, this negative value is added to 1 * multiplicand, thus calling for a second differencing operation. The various possible sign combinations are shown in Table 13.10-1.

Carry is not enabled on difference operations. Recall that, in QS, occurrence of this carry on a difference operation served to indicate which operand was larger; it was never taken to be a part of the result and never gated into S. In the case of multiply, it is known that the multiplicand is larger than the partial product since the former is binary normalized before the differencing takes place and the latter is always shifted right at least one place. Thus, it is also known that the

TABLE 13.10–1 Partial Product Sign Values

State of PS flip–flop; low for positive partial product (mode 0), high for negative (mode 1)	reset +	reset +	set −	set −
Is add or subtract operation indicated? (See Table 13.10–2 and preceding paragraph for a discussion of what constitutes add or subtract)	add	subtract	subtract	add
State of PS combined with operation = sum or difference?	sum	difference; complement partial product	sum	difference; complement partial product
New state of PS flip–flop	reset +	set −	set −	reset +
State of carry–enable flip–flop (CYE)	set	reset	set	reset

result gated from the Adder is in true form, low by 1. The CZ flip–flop is not set for differencing in QP, as it is during the operation of QS, but rather the negative partial product formed when the 1's string is entered is allowed to remain low by 1 until the end of the string. At that point, a second differencing operation is called for and the partial product is complemented. Since it was low by 1 before being complemented, it now goes high by 1. This has the same affect as would the setting of CZ and thus results in the formation of the correct final value.

Consider the multiplication of 5 by 7, ignoring for the moment the relevant exponent value. On entry into QP, the PS flip-flop is reset for mode 0. Examination of the multiplier bits (111) calls for a subtraction and, with PS reset, a differencing operation. The partial product is complemented, carry inhibited, and the partial product sign reversed to indicate both a change of mode from 0's to 1's string and a change from a positive to a negative partial product. At the end of the string, the negative partial product is added to the multiplicand, again calling for a differencing operation. The PS flip-flop is reversed, carry out inhibited, and the partial product is complemented.

The operands in this example are

multiplicand = 5 = 101

multiplier = 7 = 111

In the notation commonly used for decimal multiplication, this operation would be handled as follows:

$$\begin{array}{r} 101 \\ \underline{111} \\ 101 \\ 101 \\ \underline{101} \\ 100011 \end{array}$$

Straightforward implementation of this using the Adder circuitry would involve three add cycles:

1) multiplicand 101

   initial partial product $\underline{000}$

   101 = new partial product

2) shift partial product 1 bit to the right; increment exponent by 1/3; add:

   multiplicand 101

   partial product $\underline{101}$

   1111 = new partial product

3) shift partial product 1 bit to the right; increment exponent by 1/3; add:

$$\begin{array}{lr} \text{multiplicand} & 101 \\ \text{partial product} & \underline{1111} \\ & 100011 \end{array}$$

The number of add cycles is cut from three to two by use of the string method. The real advantage of this algorithm comes, of course, from longer strings since a string of any length can be handled by two add cycles. Using these same values, the QP sequencer would form the product as is described below.

Recall that the multiplicand has been binary normalized in register N and that the S register has been cleared to zero for the initial partial product value. Thus, the Adder input registers are set up as follows:

$$\text{multiplicand} = (N)\ \overset{41\qquad\qquad\qquad\qquad 0}{\boxed{1010\ .\ .\ .\ .\ .\ .\ .\ .\ .\ 0}}$$

$$\text{initial partial product} = (D)\ \overset{41\qquad\qquad\qquad\qquad 0}{\boxed{0000\ .\ .\ .\ .\ .\ .\ .\ .\ .\ 0}}$$

The multiplier is stored in register A.

$$\text{multiplier} = (A)\ \overset{41\qquad\qquad\qquad\qquad 0}{\boxed{0000\ .\ .\ .\ .\ .\ .\ .\ .\ .\ 0111}}$$

Examination of the 3 least significant bits indicates that a 1's string is being entered. The multiplier and the partial product are shifted right 3 since all the bits in the multiplier are 1's. The indicated subtract operation, with PS reset, calls for a differencing operation. Thus, the initial partial product is complemented while being shifted so that

$$(D) = \overset{41\qquad\qquad\qquad\qquad 0}{\boxed{1111\ .\ .\ .\ .\ .\ .\ .\ .\ .\ 1}}$$

The inputs to the Adder are gated to S and the octal exponent is incremented by 1.

multiplicand = (N) = 101000000000000000000000000000000000000000

partial product = (D) = $\underline{111111111111111111111111111111111111111111}$

new partial product = (S) = 100111111111111111111111111111111111111111

(Carry is not enabled for differencing operations.) The right 3 shift of the multiplier brings three zeros into the least significant 3 bits of register A. This indicates the end of the string and that an add cycle is in order. A negative partial product and an addition call for a second differencing operation. The multiplier is shifted right 3 bits while the partial product is complemented and shifted right 3 so that

$$(D) = \overset{41\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad 0}{\boxed{111011000000000000000000000000000000000000}}$$

The second add cycle then occurs as follows:

multiplicand = (N) = 101000000000000000000000000000000000000000

partial product = (D) = $\underline{111011000000000000000000000000000000000000}$

final product = (S) = 100011000000000000000000000000000000000000

The octal exponent is incremented by 1. Thus, the correct result was obtained through the use of two add cycles.

This discussion should clarify the action taken when the least significant 3 multiplier bits are all zeros or all ones. It is also obvious that a zeros string broken by a single 1 bit calls for an addition at that order. (Mode 0 and 001 indicate add and shift right 3.) It is easy to demonstrate that the opposite case, mode 1 and 110, is handled analogously, that is, a subtraction occurs at the order where the zero appears in the multiplier and shifting continues. This is true since, for example, 110111 = 1000000 – (1 + 001000)

so that the multiplication of 5 (101) by 67_8 (110111) is equivalent to

1000000 X 101 – 1 X 101 – 1000 X 101.

Using decimal format, this multiplication would appear as follows:

```
      101
   110111
   ------
      101
     101
    101
   000
  101
 101
---------
100010011
```

This operation, as handled by QP, begins with the operands established in N and D:

```
                                   41              0
multiplicand = (N) =               |1010 . . . . . . . . . 0|

                                           41              0
initial partial product = (D) =            |0000 . . . . . . . . . 0|
```

The multiplier is in register A.

```
                        41                   0
multiplier = (A) =      |0000 . . . . . . . . . 110111|
```

Examination of the 3 least significant bits in register A indicates entry into a 1's string (since PS indicates mode 0). The multiplier is shifted right 3, the partial product is complemented and shifted right 3. The Adder is then gated to S and the octal exponent incremented by 1.

```
multiplicand    = (N) = 101000000000000000000000000000000000000000
partial product = (D) = 111111111111111111111111111111111111111111
                        ------------------------------------------
new partial     = (S) = 100111111111111111111111111111111111111111
product
```

Carry is not enabled for differencing operations. The least significant 3 bits in the multiplier are now 110 and the PS flip–flop indicates mode 1. It is not necessary to leave the string on account of the zero; instead, a subtraction is performed at that order and, the other bits being 1's, shifts of right 3 take place. Referring to Table 13.10–1, the subtraction when PS is set (negative) calls for summing. PS will remain set and carry out will be enabled to avoid loss of significance. The shift of 3 performed on the partial product establishes (D) as follows:

```
        41                                         0
(D) = |000100111111111111111111111111111111111111|
```

The Adder is gated to S and the exponent incremented by 1.

```
multiplicand    = (N) = 101000000000000000000000000000000000000000
partial product = (D) = 000100111111111111111111111111111111111111
new partial     = (S) = 101100111111111111111111111111111111111111
product
```

The last shift of the multiplier brought a string of zeros into the least significant 3 bits. This calls for a change of mode through the addition of the negative partial product (differencing operation). The multiplier is shifted right 3, the partial product in S is complemented and shifted right 3 to D.

```
        41                                         0
(D) = |111010011000000000000000000000000000000000|
```

The final add cycle is gated to S. Since this is a differencing operation, carry out is not enabled.

```
multiplicand    = (N) = 101000000000000000000000000000000000000000
partial product = (D) = 111010011000000000000000000000000000000000
final product   = (S) = 100010011000000000000000000000000000000000
```

All other cases involve permutations of those that have been described. Thus, bits 010 in mode 0 call for a right shift of 1 so that the addition can take place at that order, while bits 101 in mode 1 call for a right shift of 1.

There are 4 distinct cases calling for use of the Adder, each of which results from two possible bit configurations. (The other instances call for shifting but no add cycle.) A summing or differencing operation takes place on the basis of the combination of the sign in the PS flip–flop and the add or subtract sign. The decision to add or subtract is based on what is happening in the current string. Subtraction is indicated when a 1's string is entered, addition when it is left. The occurrence of a single 1 in a string of zeros calls for addition, that of a single 0 in a string of ones for subtraction. Thus, summing occurs if the sign held in the PS flip–flop agrees with that of the operation called for, i.e., PS high, indicating negative, and a subtraction result in the summing of the operands. These factors and their results are shown in Table 13.10–2.

TABLE 13.10–2 Determination of Summing or Differencing Operations

	Subtract			Add		
	Bits	Meaning	Operation	Bits	Meaning	Operation
Mode 0 $\overline{\text{PS}}$	111 or 011	Entry into 1's string	Differencing	010 or 101	Single 1 in 0's string	Summing
Mode 1 PS	110 or 010	Single 0 in 1's string	Summing	000 or 100	Entry into 0's string	Differencing

From this it is clear that differencing occurs only on entry into a 1's string or a 0's string. (It could equally well be stated that differencing occurs only upon entry into or exit from a 1's string.)

All of the possible cases and the actions taken are shown in Table 13.10–3. The signals generated by the QP sequencer on examination of these 3 bits are shown in the table as an aid in relating the general rule to the implementation as carried out by QP. These signals have the following meanings:

1) SHT – SHift right Three;
2) SHN – SHift right oNe;
3) AST – Add, Shift right Three;
4) ASW – Add, Shift right tWo.

Note that the cases where right shifts of 2 bits are called for cause signal SHN to go high whereas the cases where right shifts of 2 bits occur in conjunction with add cycles cause ASW to go high. This is due to the fact that it is possible to achieve a right–2–bit shift when the Adder is used (enabling of S(R3)D followed by gating from the Adder using R(L1)S accomplishes this) while there is no direct means for performing a right–2–bit shift without the Adder. Thus, SHN is used twice in this case and the 2–bit shift results from two single shifts.

The QP sequencer performs three distinct operations: initialization, product formation and cleanup. Initialization involves states QPA, QPB, and QPC. Actions taken here include the clearing of the initial partial product, resetting of the PS flip–flop, inhibiting of the add cycle to correctly direct the branching on entry into QPD, etc.

Product formation is carried out by the QPD and QPE states. The sequencer does not proceed from QPE until the last multiplier bit has

TABLE 13.10–3 Actions Taken Following Examination of the Least Significant Three Bits of the Multiplier

MODE 0; PS RESET FOR POSITIVE SIGN						MULT. BITS	MODE 1; PS SET FOR NEGATIVE SIGN					
Shift	Add/Sub	Sum/Dif	PS	CYE	Signal		Shift	Add/Sub	Sum/Dif	PS	CYE	Signal
R3	—	—	—	—	SHT	000	R3	Add	Dif	Reset	Reset	AST
R2	—	—	—	—	SHN	100	R2	Add	Dif	Reset	Reset	ASW
R1	—	—	—	—	SHN	010	R2	Sub	Sum	Set	Set	ASW
R1	—	—	—	—	SHN	110	R3	Sub	Sum	Set	Set	AST
R3	Add	Sum	Reset	Set	AST	001	R1	—	—	—	—	SHN
R2	Add	Sum	Reset	Set	ASW	101	R1	—	—	—	—	SHN
R2	Sub	Dif	Set	Reset	ASW	011	R2	—	—	—	—	SHN
R3	Sub	Dif	Set	Reset	AST	111	R3	—	—	—	—	SHT

All shifts refer to the partial product and to the multiplier; the product exponent is incremented in step. The partial product is complemented before each differencing operation.

been examined. At QPD the multiplier is in register A and the bits examined are A2, A1 and A0. If no add cycle is called for at this time, the decision is whether or not a differencing operation (a change of mode) is called for. This amounts to picking out four cases and complementing the partial product for each; $\overline{PS} \wedge 011$, $\overline{PS} \wedge 111$, $PS \wedge 100$ and $PS \wedge 000$. Only the 2 least significant bits are essential to this determination. Consequently, the CMP or complement path is taken if $\overline{PS} \wedge 11$ or if $PS \wedge 00$. Examination of A2, A1 and A0 will set SHN, SHT, ASW or AST to determine the path taken at QPE. At QPE, the bits that were positioned in A2, A1 and A0 will be in M–1, M–2 and M–3. (This regardless of what total shift is required; a right 3 shift always occurs first; sometimes it is erased by not copying it back, sometimes it is modified by the enabling of the left–2 path as it is sent back to A.) For the ASW case, SHW is set at QPE to cause a branch during the add cycle that will correctly complete the 2–bit shift. The paths taken between QPD and QPE are described in Table 13.10–4. Each of these is demonstrated in Figure 13.10–1.

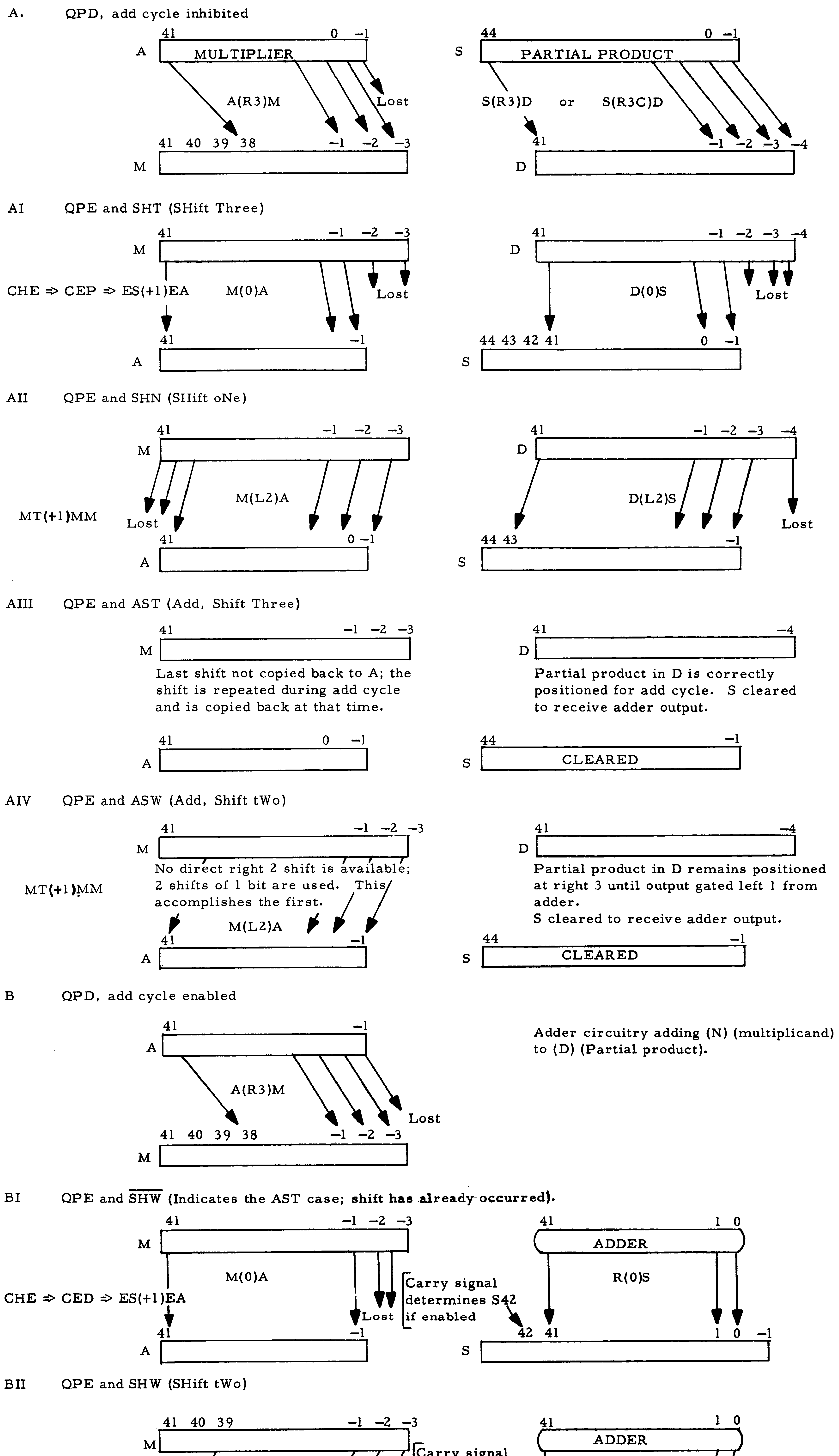
The cleanup phase of the sequencer includes states QPF, QPG and QPH. In this phase, the product is shifted until it is octally justified, that is, the exponent value is a whole number. Due to the fact that QPD and QPE represent a loop, the usual rules for establishing decoding states have been ignored. QPD is on clock 2 and would normally be decoded by the echo of QPC. Instead, it is decoded by the echo of QPE, while QPH is the clock 2 state with the echo decoding for QPC. Thus, at QPC, the QP2 and QP4 flip–flops are set to advance the sequencer to QPD.

The algorithm for the entire QP sequencer is shown in Figure 13.10–2 and the QP flow chart in Figure 13.10–3.

TABLE 13.10-4 Paths in Product Formation States of QP: QPD and QPE

STATE:	BRANCHES:	BRANCHES:
QPD	A. Add cycle inhibited (If change of mode called for by $PS \wedge \overline{A0} \wedge \overline{A1} \vee \overline{PS} \wedge A0 \wedge A1$, differencing operation is set up by complementing the partial product and inhibiting adder carry.)	B. Add cycle enabled This assumes that more than 1 multiplier bit remains. Otherwise, cleanup phase (QPF, QPG, QPH) is entered.
QPE	Four branches occur based on A above:	Two branches occur based on B above:
	I. SHT (SHift Three); increment exponent by 1. Indicates that current string continues: Mode $0 \wedge 000$ or Mode $1 \wedge 111$ II. SHN (SHift oNe); increment exponent by 1/3. Indicates that current string ends or is interrupted after the next 1 or 2 bits: Mode $0 \wedge 100 \vee 010 \vee 110$ or Mode $1 \wedge 001 \vee 101 \vee 011$ III. AST (Add, Shift Three); Indicates entry into or exit from a string calling for differencing operation: Mode $0 \wedge 111$ or Mode $1 \wedge 000$ or interruption of a string calling for a subtraction if Mode $1 \wedge 110$, an addition if Mode $0 \wedge 001$. Enable add cycle. IV. ASW (Add, Shift tWo); increment exponent by 1/3. Indicates exit from a string calling for differencing operation: Mode $0 \wedge 011$ or Mode $1 \wedge 100$ or interruption of a string calling for a subtraction if Mode $1 \wedge 010$, an addition if Mode $0 \wedge 101$. Enable add cycle, set SHW.	I.* SHW (SHift tWo); increment exponent by 1/3. Adder gated left 1 to complete partial product shift. Two right-1-bit shifts necessary between M and A since no direct means available for 2-bit shifts. II.* $\overline{SHW}$; increment exponent by 1. Adder gated to S; 3-bit shift has been completed. *For both cases, reverse PS if change of mode is indicated (differencing operation).

FIGURE 13.10–1 Demonstration of Paths Between QPD and QPE



Last shift not copied back to A; the shift is repeated during add cycle and is copied back at that time.

Partial product in D is correctly positioned for add cycle. S cleared to receive adder output.

No direct right 2 shift is available; 2 shifts of 1 bit are used. This accomplishes the first.

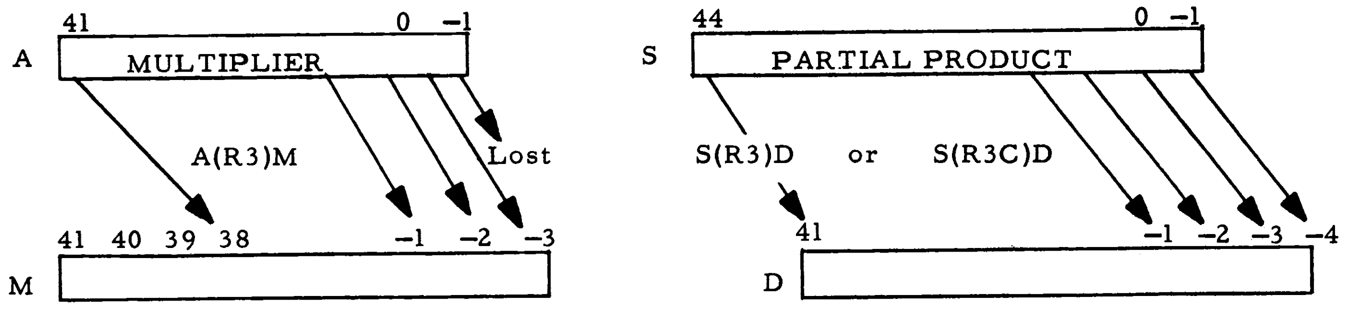
Partial product in D remains positioned at right 3 until output gated left 1 from adder.
S cleared to receive adder output.

Adder circuitry adding (N) (multiplicand) to (D) (Partial product).

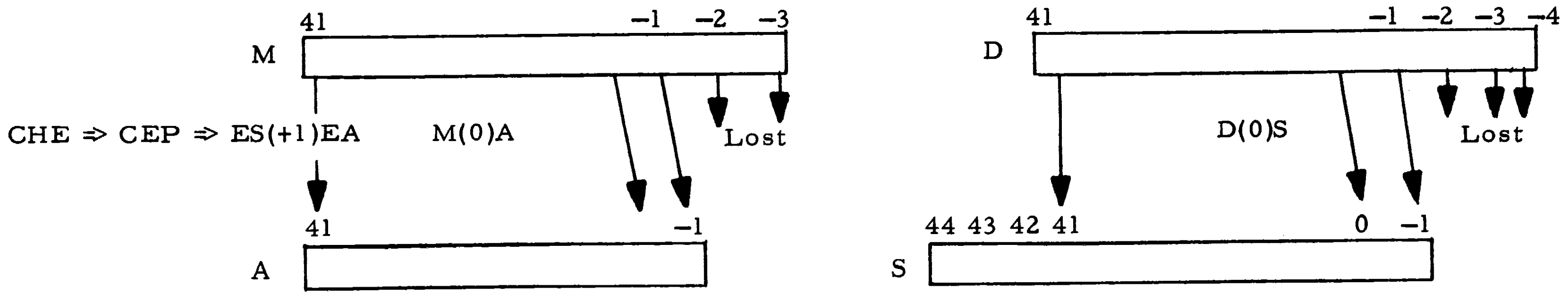
Carry signal determines S42 if enabled

Carry signal

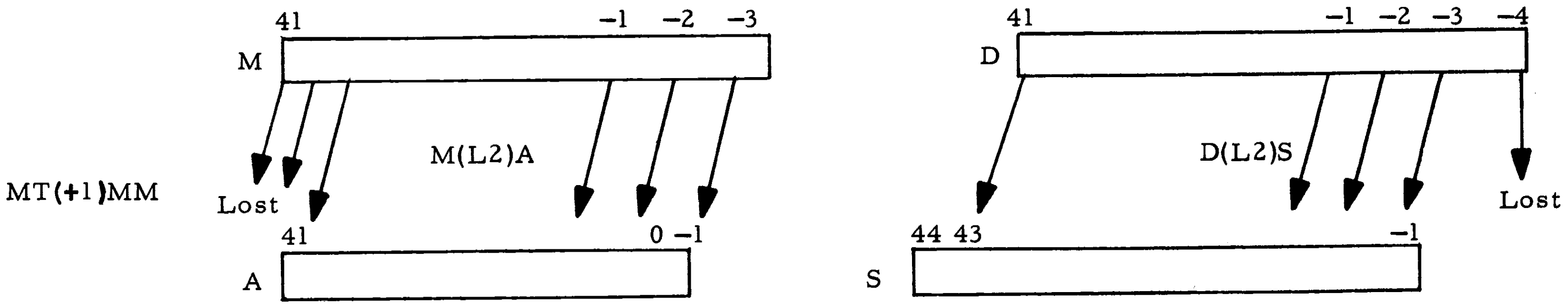
A. QPD, add cycle inhibited



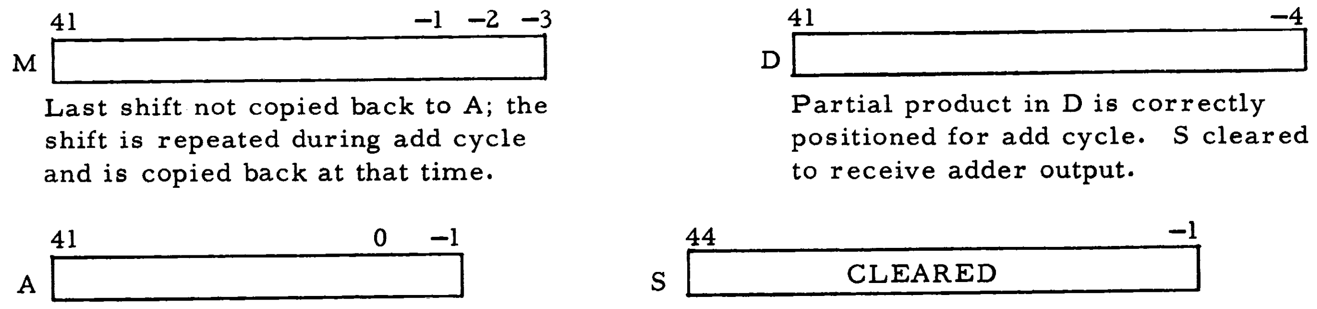
AI QPE and SHT (SHift Three)



AII QPE and SHN (SHift oNe)



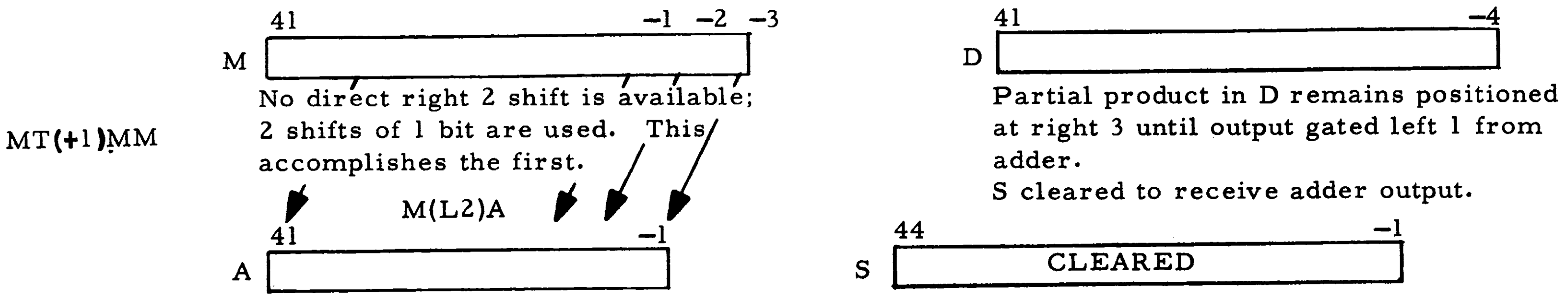
AIII QPE and AST (Add, Shift Three)



Last shift not copied back to A; the shift is repeated during add cycle and is copied back at that time.

Partial product in D is correctly positioned for add cycle. S cleared to receive adder output.

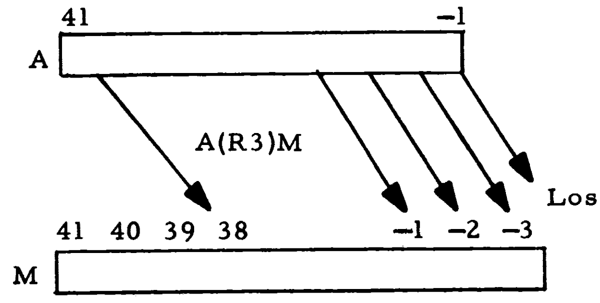
AIV QPE and ASW (Add, Shift tWo)



No direct right 2 shift is available; 2 shifts of 1 bit are used. This accomplishes the first.

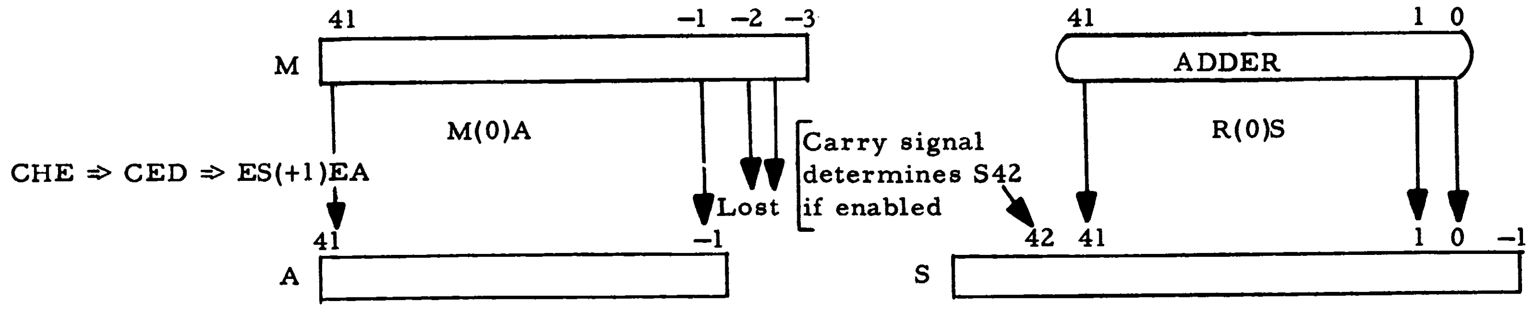
Partial product in D remains positioned at right 3 until output gated left 1 from adder.
S cleared to receive adder output.

B QPD, add cycle enabled



Adder circuitry adding (N) (multiplicand) to (D) (Partial product).

BI QPE and $\overline{\text{SHW}}$ (Indicates the AST case; shift has already occurred).



BII QPE and SHW (SHift tWo)

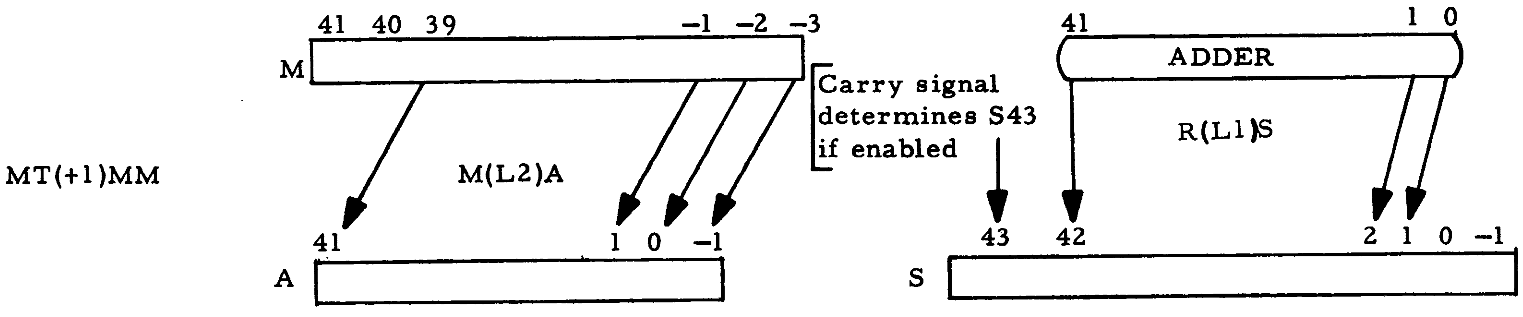


FIGURE 13.10–2 Algorithm for Product Formation by QP

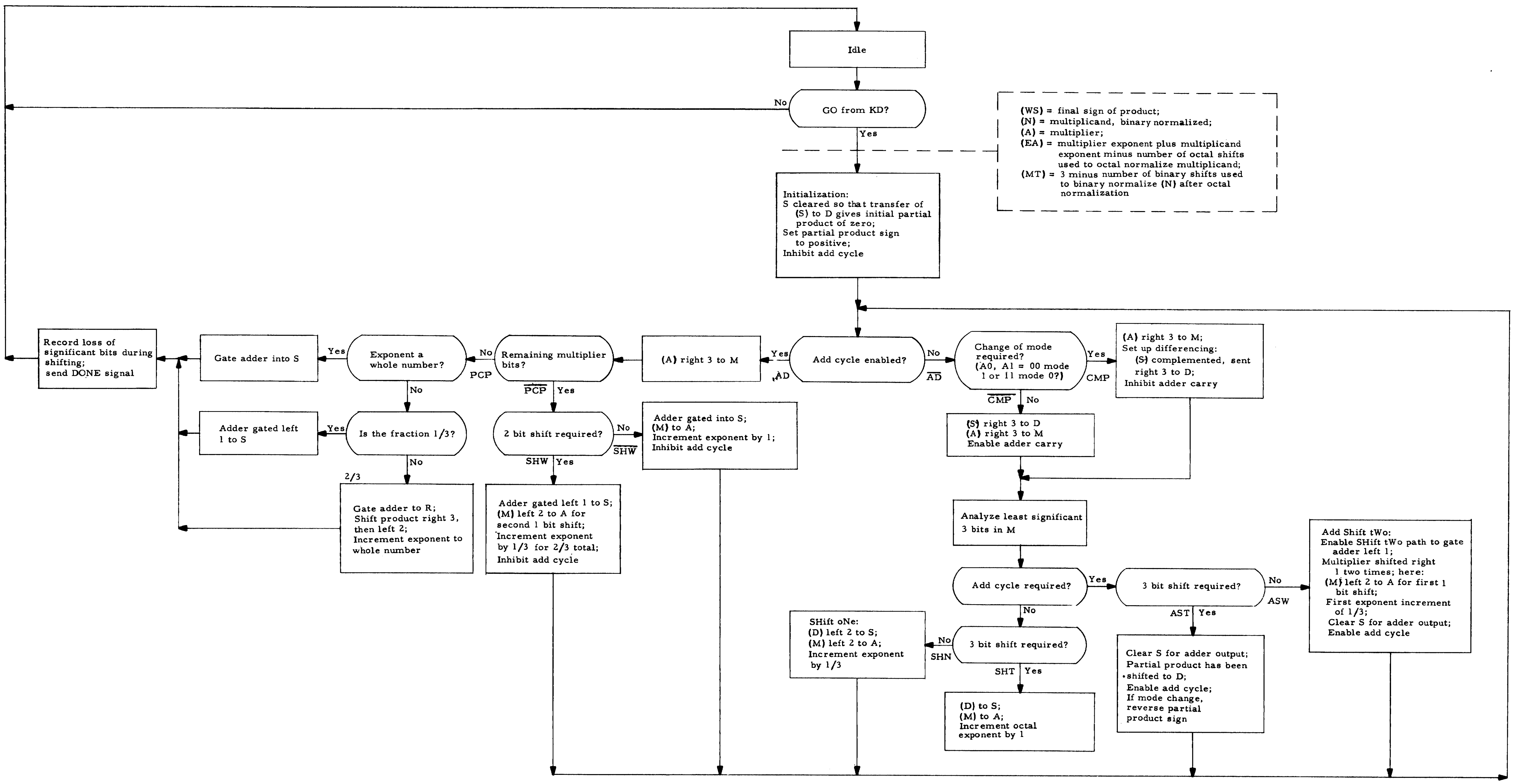
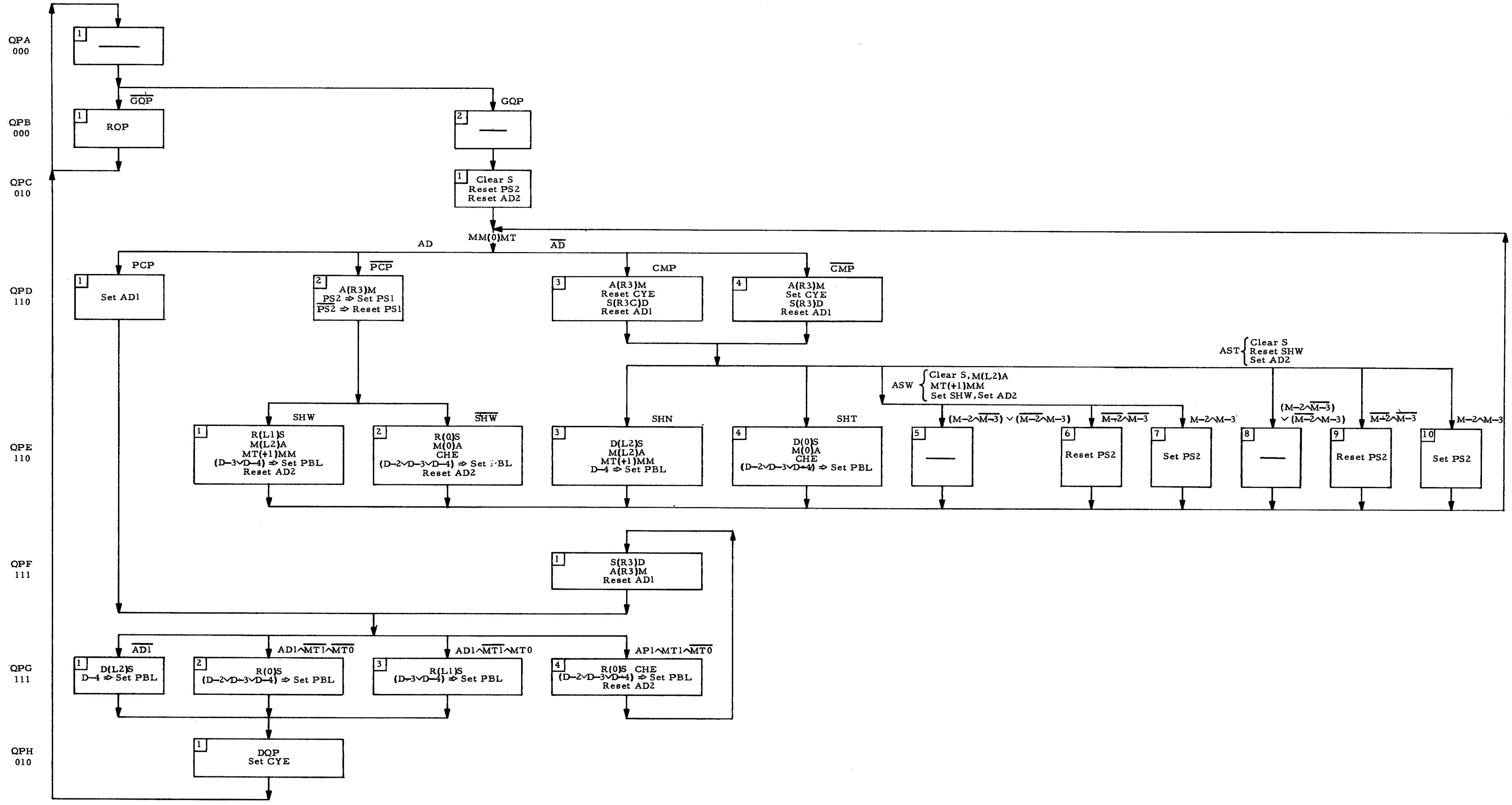

Idle
GO from KD?
No
Yes
(WS) = final sign of product;
(N) = multiplicand, binary normalized;
(A) = multiplier;
(EA) = multiplier exponent plus multiplicand exponent minus number of octal shifts used to octal normalize multiplicand;
(MT) = 3 minus number of binary shifts used to binary normalize (N) after octal normalization
Initialization:
S cleared so that transfer of (S) to D gives initial partial product of zero;
Set partial product sign to positive;
Inhibit add cycle
Add cycle enabled?
Yes
AD
No
$\overline{AD}$
(A) right 3 to M
Remaining multiplier bits?
No
PCP
$\overline{PCP}$
Yes
Exponent a whole number?
Yes
No
Gate adder into S
Record loss of significant bits during shifting; send DONE signal
Is the fraction 1/3?
Yes
No
2/3
Adder gated left 1 to S
Gate adder to R; Shift product right 3, then left 2; Increment exponent to whole number
2 bit shift required?
No
$\overline{SHW}$
SHW
Yes
Adder gated into S; (M) to A; Increment exponent by 1; Inhibit add cycle
Adder gated left 1 to S; (M) left 2 to A for second 1 bit shift; Increment exponent by 1/3 for 2/3 total; Inhibit add cycle
Change of mode required? (A0, A1 = 00 mode 1 or 11 mode 0?)
Yes
CMP
$\overline{CMP}$
No
(A) right 3 to M; Set up differencing: (S) complemented, sent right 3 to D; Inhibit adder carry
(S) right 3 to D
(A) right 3 to M
Enable adder carry
Analyze least significant 3 bits in M
Add cycle required?
Yes
No
3 bit shift required?
No
ASW
AST
Yes
Add Shift tWo: Enable SHift tWo path to gate adder left 1; Multiplier shifted right 1 two times; here: (M) left 2 to A for first 1 bit shift; First exponent increment of 1/3; Clear S for adder output; Enable add cycle
Clear S for adder output; Partial product has been shifted to D; Enable add cycle; If mode change, reverse partial product sign
3 bit shift required?
No
SHN
SHT
Yes
SHift oNe: (D) left 2 to S; (M) left 2 to A; Increment exponent by 1/3
(D) to S; (M) to A; Increment octal exponent by 1

TABLE 13.10–5 Terms Used on the QP Flow Chart

AD1, AD2	Add Delay flip–flops; AD2 is set during multiply if ADD is high at state QPE and AD1 is low.
ADD	ADD cycle is indicated; ADD is high when the states of the least significant multiplier bit and the partial product sign are opposite one another; hence, ADD = $PS1 \wedge \overline{M-3} \vee \overline{PS1} \wedge M-3$.
AST	Add and Shift right Three bits; AST = $PS1 \wedge [(M-1 \wedge M-2 \wedge \overline{M-3}) \vee (\overline{M-1} \wedge \overline{M-2} \wedge \overline{M-3})] \vee \overline{PS1} \wedge [(\overline{M-1} \wedge \overline{M-2} \wedge M-3) \vee (M-1 \wedge M-2 \wedge M-3)]$.
ASW	ADD and Shift right tWo bits; ASW = $PS1 \wedge [(M-1 \wedge \overline{M-2} \wedge \overline{M-3}) \vee (\overline{M-1} \wedge M-2 \wedge \overline{M-3})] \vee \overline{PS1} \wedge [(\overline{M-1} \wedge M-2 \wedge M-3) \vee (M-1 \wedge \overline{M-2} \wedge M-3)]$; ASW high causes a shift of right 1 and sets SHW so that an additional 1–bit shift will occur at QPE on the add cycle.
CHE	CHange octal Exponent signal.
CMP	CoMPlement inverter; a change of mode sets this flip–flop which causes the partial product to be complemented; thus, CMP = $(PS2 \wedge \overline{A1} \wedge \overline{A0}) \vee (\overline{PS2} \wedge A1 \wedge A0)$ since the mode is changed when the sign is negative and the least significant 2 bits in the multiplier are 0 and when the sign is positive and these bits are 1.
CYE	CarrY Enable flip–flop; CYE is high to enable a carry into the 42nd or 43rd bit of the S register (depending upon the path used to gate the output); during the operation of QP, CYE is set at the QPD state if this is not an add cycle and CMP is low and at the QPH cycle on leaving QP; CYE is reset at QPD if this is not an add cycle and CMP is high.
$DQP \vee RQP$	DONE QP sequencer or READY QP sequencer; these signals are equivalent to $\overline{QP2}$ since QP2 is always high while QP is in operation.
EA(0)ES	Path enabled on all C1's while QP is in operation since KD is idling at KDG–KDH where this enable occurs. This updates the octal exponent counter.
ES(+1)EA	Path enabled to cause octal increment of product exponent; this path is enabled on C2's if $CHE \vee (QP1 \wedge MT(+1)MM \wedge MT1 \wedge \overline{MT0})$.
GQP	GO QP sequencer; GQP = $KDF \wedge QPB \wedge \overline{AZ} \wedge S44 \wedge CD10 \wedge \overline{SZ}$.
$M-1 \wedge M-2 \wedge M-3$	Three least significant multiplier bits; these same bits are referred to as $A2 \wedge A1 \wedge A0$ when the multiplier is in register A since the multiplier is shifted 3 bits to the right when it goes to M.
PBL	Product Bits Lost flip–flop; PBL is set when significant bits are shifted out of the D register; PBL is taken into account when the decision is made to round the product or not.
PCP	Product Cleanup Phase; PCP is high when at most 1 multiplier bit remains in register A. There are only 2 possible termination cases: if mode 0, the procedure is to add and shift right 3; if mode 1, to complement, add and shift right 3. PCP = $41AZ1 \wedge \overline{PS}$.
PS1, PS2	Partial product Sign flip–flop; PS is low for positive, high for negative values. Set PS2 = $QPE \wedge \overline{AD1} \wedge M-2 \wedge M-3$; reset PS2 = $QPE \wedge \overline{AD1} \wedge \overline{M-2} \wedge \overline{M-3} \vee QPC$
SHN	SHift right oNe bit; SHN is high if $PS1 \wedge [(\overline{M-2} \wedge M-3) \vee (\overline{M-1} \wedge M-3)] \vee \overline{PS1} \wedge [(M-2 \wedge \overline{M-3}) \vee (M-1 \wedge \overline{M-3})]$.
SHT	SHift right Three bits; SHT is high if $(PS1 \wedge M-1 \wedge M-2 \wedge M-3) \vee (\overline{PS1} \wedge \overline{M-1} \wedge \overline{M-2} \wedge \overline{M-3})$.
SHW	SHift right tWo bits flip–flop; SHW is set at QPE if an add cycle and 2 bit shift are called for (ASW high); ASW causes a 1–bit shift; SHW gates another 1–bit shift to complete the 2–bit shift; (there is no direct 2–bit shift path).

FIGURE 13.10-3 The QP Flow Chart

QP


QPA 000
QPB 000
QPC 010
QPD 110
QPE 110
QPF 111
QPG 111
QPH 010
1 —
$\overline{GQP}$
GQP
1 RQP
2 —
1 Clear S Reset PS2 Reset AD2
MM(0)MT
AD
$\overline{AD}$
PCP
$\overline{PCP}$
CMP
$\overline{CMP}$
1 Set AD1
2 A(R3)M PS2 ⇒ Set PS1 $\overline{PS2}$ ⇒ Reset PS1
3 A(R3)M Reset CYE S(R3C)D Reset AD1
4 A(R3)M Set CYE S(R3)D Reset AD1
AST {Clear S Reset SHW Set AD2
ASW {Clear S, M(L2)A MT(+1)MM Set SHW, Set AD2
SHW
$\overline{SHW}$
SHN
SHT
$(M{-}2\wedge\overline{M{-}3})\vee(\overline{M{-}2}\wedge M{-}3)$
$\overline{M{-}2}\wedge\overline{M{-}3}$
$M{-}2\wedge M{-}3$
$(M{-}2\wedge\overline{M{-}3})\vee(\overline{M{-}2}\wedge M{-}3)$
$\overline{M{-}2}\wedge\overline{M{-}3}$
$M{-}2\wedge M{-}3$
1 R(L1)S M(L2)A MT(+1)MM (D−3∨D−4) ⇒ Set PBL Reset AD2
2 R(0)S M(0)A CHE (D−2∨D−3∨D−4) ⇒ Set PBL Reset AD2
3 D(L2)S M(L2)A MT(+1)MM D−4 ⇒ Set PBL
4 D(0)S M(0)A CHE (D−2∨D−3∨D−4) ⇒ Set PBL
5 —
6 Reset PS2
7 Set PS2
8 —
9 Reset PS2
10 Set PS2
1 S(R3)D A(R3)M Reset AD1
$\overline{AD1}$
$AD1\wedge\overline{MT1}\wedge\overline{MT0}$
$AD1\wedge\overline{MT1}\wedge MT0$
$AP1\wedge MT1\wedge\overline{MT0}$
1 D(L2)S D−4 ⇒ Set PBL
2 R(0)S (D−2∨D−3∨D−4) ⇒ Set PBL
3 R(L1)S (D−3∨D−4) ⇒ Set PBL
4 R(0)S CHE (D−2∨D−3∨D−4) ⇒ Set PBL Reset AD2
1 DQP Set CYE

SECTION 13.11 – KD SEQUENCER CONTROL OF DIVIDE AND REVERSE DIVIDE

Quotient formation during division is accomplished by the QQ sequencer. Before QQ is started, the divide operation is set up by the QA and KD sequencers. The interaction between these sequencers is shown in Figure 13.11–1. KD and QA are started simultaneously by KC; KD waits until QA has reached the QAH state before proceeding. At this point QA will have accomplished the following:

1) Formation of the initial quotient exponent. For DIV, this exponent value = (EA) – (EP); for RDV, the exponent is (EP) – (EA). QA stores this exponent in register EA, with the attached sign in EA8.

2) Determination of quotient sign and storage in the WS flip–flop.

3) Switching of (A) with (N) for RDV command so that, when QA is finished, in all cases (A) = numerator, (N) = denominator.

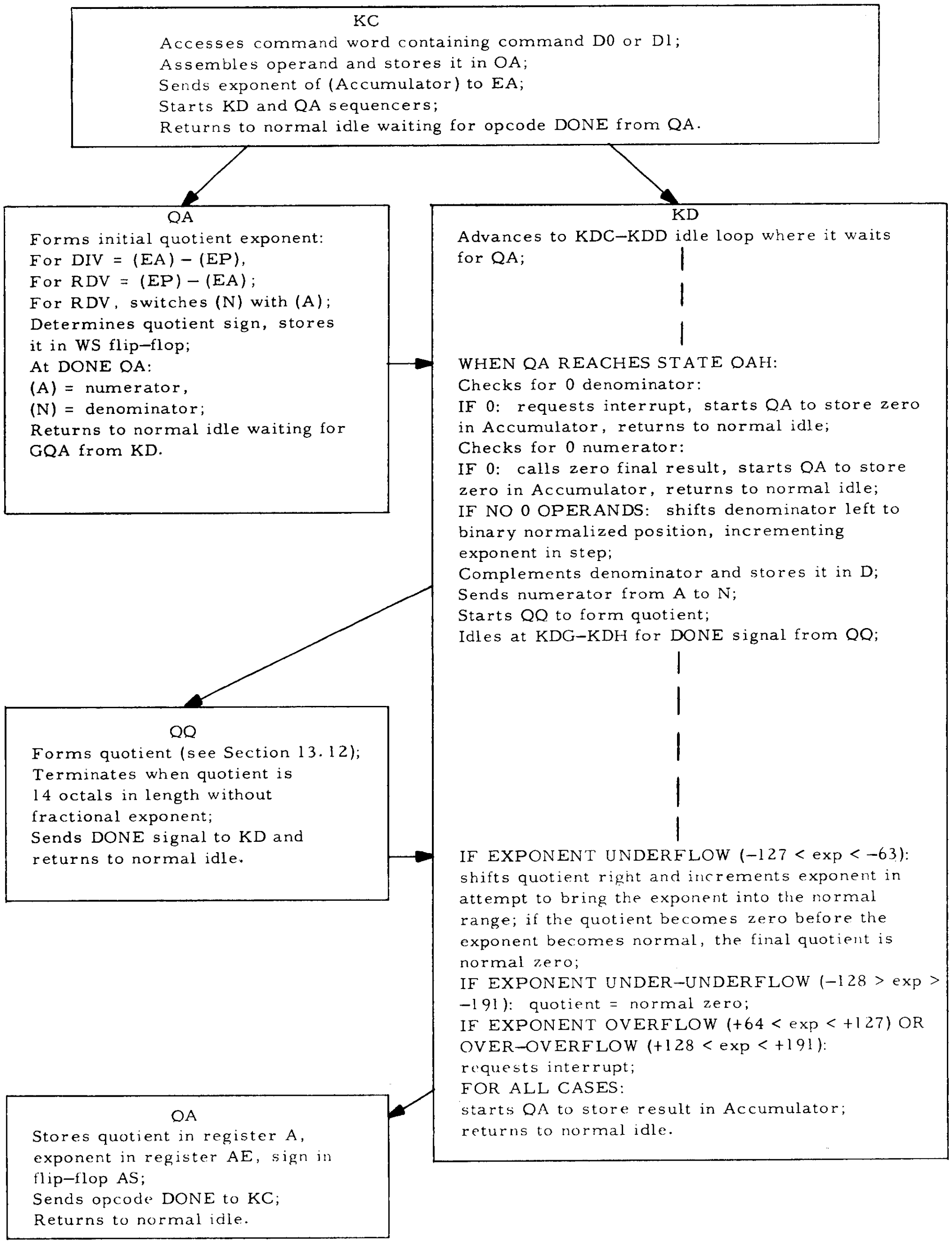
Incrementing and decrementing of this initial quotient value during the setting up of the divide operation requires some explanation. The relevant manipulations are as follows:

1) The exponent of the denominator e_d, is subtracted from that of the numerator, e_n, to become the preliminary quotient exponent, e_q; thus, $e_n - e_d = e_q$.

2) The denominator is binary normalized. As the denominator is shifted left, the exponent originally associated with the denominator, e_d, would become smaller; however, the subtraction described in 1) above has already occurred and the result of decreasing the size of e_d is to increase the size of e_q; therefore, e_q is incremented in step with the left shifts of the denominator.

FIGURE 13.11-1 General Algorithm for Divide and Reverse Divide Operations

D0 (DIV) calls for use of the operand from the command word as the denominator. (This operand will be in QA at the end of operand assembly.) The operand stored in the Accumulator is the numerator for this operation. D1 (RDV) calls for reversal of the roles of the operands: that in QA is the numerator, that in the Accumulator, the denominator. Processing is as follows:



KC
Accesses command word containing command D0 or D1;
Assembles operand and stores it in QA;
Sends exponent of (Accumulator) to EA;
Starts KD and QA sequencers;
Returns to normal idle waiting for opcode DONE from QA.

QA
Forms initial quotient exponent:
For DIV = (EA) – (EP),
For RDV = (EP) – (EA);
For RDV, switches (N) with (A);
Determines quotient sign, stores it in WS flip–flop;
At DONE QA:
(A) = numerator,
(N) = denominator;
Returns to normal idle waiting for GQA from KD.

KD
Advances to KDC–KDD idle loop where it waits for QA;

WHEN QA REACHES STATE QAH:
Checks for 0 denominator:
IF 0: requests interrupt, starts QA to store zero in Accumulator, returns to normal idle;
Checks for 0 numerator:
IF 0: calls zero final result, starts QA to store zero in Accumulator, returns to normal idle;
IF NO 0 OPERANDS: shifts denominator left to binary normalized position, incrementing exponent in step;
Complements denominator and stores it in D;
Sends numerator from A to N;
Starts QQ to form quotient;
Idles at KDG–KDH for DONE signal from QQ;

IF EXPONENT UNDERFLOW (–127 < exp < –63): shifts quotient right and increments exponent in attempt to bring the exponent into the normal range; if the quotient becomes zero before the exponent becomes normal, the final quotient is normal zero;
IF EXPONENT UNDER–UNDERFLOW (–128 > exp > –191): quotient = normal zero;
IF EXPONENT OVERFLOW (+64 < exp < +127) OR OVER–OVERFLOW (+128 < exp < +191): requests interrupt;
FOR ALL CASES:
starts QA to store result in Accumulator;
returns to normal idle.

QQ
Forms quotient (see Section 13.12);
Terminates when quotient is 14 octals in length without fractional exponent;
Sends DONE signal to KD and returns to normal idle.

QA
Stores quotient in register A, exponent in register AE, sign in flip–flop AS;
Sends opcode DONE to KC;
Returns to normal idle.

3) During the operation of QQ, the numerator (or remainder) is binary normalized before each use of the Adder. The affect of the left shifts performed on the numerator is a decrease in the size of its exponent value, e_n, and also a decrease in the size of the quotient exponent, e_q; thus, QQ decrements e_q in step with the normalization of the numerator.

Binary normalization is achieved by means of octal shifts to the left, with exponent changes of 1, until octal normalization occurs, followed by binary shifts, with exponent changes of 1/3 (handled by the modulo–three counter), until binary normalization occurs.

The KD sequencer binary normalizes the denominator, then complements it and stores it in register D where it remains until the division is complete. Note that complementation of the binary normalized value means that the most significant bit in D will always be a zero. KD requests an interrupt if division by zero is attempted and supplies a normal zero result if division into zero is attempted. When QQ completes formation of the 14 octal quotient, KD checks the exponent range. An overflow results in an interrupt, an under–underflow in a zero result. If an underflow condition exists, KD will attempt to bring the exponent into normal range by means of right shifts combined with exponent increments. If the quotient becomes zero before the exponent becomes normal, the result is a normal zero. No rounding takes place in divide; the quotient is truncated to 14 octals. At completion, KD starts QA to store the result in the Accumulator and returns to normal idle.

Figure 13.11–2 shows the KD algorithm for the control of divide operations. Figure 13.11–3 is the corresponding flow chart while Figure 13.11–4 is the entire KD flow chart, included here for the sake of completeness.

FIGURE 13.11–2 Algorithm for KD Control of Divide and Reverse Divide Operations

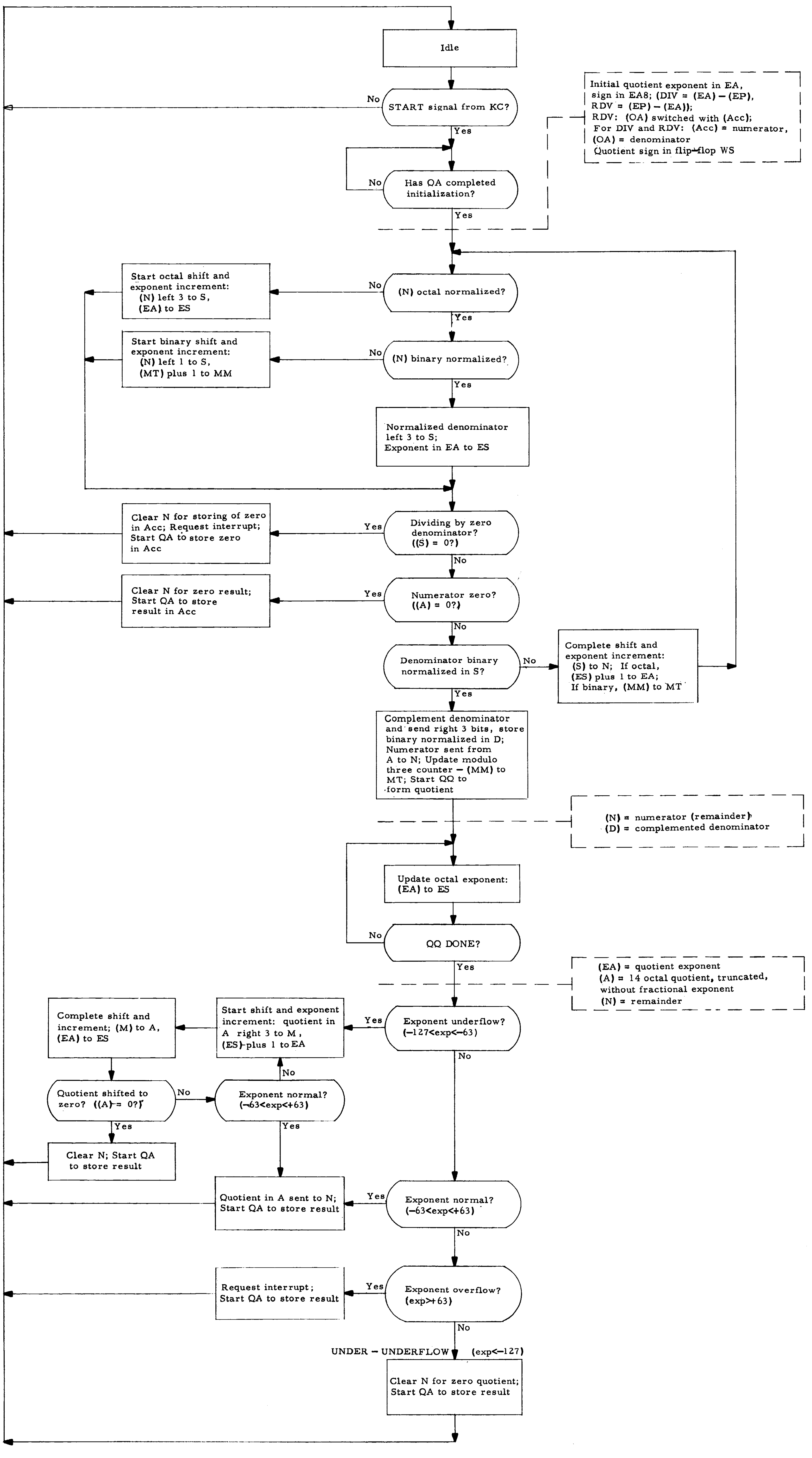

Idle
No
START signal from KC?
Yes
Initial quotient exponent in EA, sign in EA8; (DIV = (EA) – (EP), RDV = (EP) – (EA)); RDV: (OA) switched with (Acc); For DIV and RDV: (Acc) = numerator, (OA) = denominator Quotient sign in flip-flop WS
No
Has QA completed initialization?
Yes
Start octal shift and exponent increment: (N) left 3 to S, (EA) to ES
No
(N) octal normalized?
Yes
Start binary shift and exponent increment: (N) left 1 to S, (MT) plus 1 to MM
No
(N) binary normalized?
Yes
Normalized denominator left 3 to S; Exponent in EA to ES
Clear N for storing of zero in Acc; Request interrupt; Start QA to store zero in Acc
Yes
Dividing by zero denominator? ((S) = 0?)
No
Clear N for zero result; Start QA to store result in Acc
Yes
Numerator zero? ((A) = 0?)
No
Denominator binary normalized in S?
No
Complete shift and exponent increment: (S) to N; If octal, (ES) plus 1 to EA; If binary, (MM) to MT
Yes
Complement denominator and send right 3 bits, store binary normalized in D; Numerator sent from A to N; Update modulo three counter – (MM) to MT; Start QQ to form quotient
(N) = numerator (remainder) (D) = complemented denominator
Update octal exponent: (EA) to ES
No
QQ DONE?
Yes
(EA) = quotient exponent (A) = 14 octal quotient, truncated, without fractional exponent (N) = remainder
Complete shift and increment; (M) to A, (EA) to ES
Start shift and exponent increment: quotient in A right 3 to M, (ES) plus 1 to EA
Yes
Exponent underflow? (–127<exp<–63)
No
No
Quotient shifted to zero? ((A) = 0?)
No
Exponent normal? (–63<exp<+63)
Yes
Yes
Clear N; Start QA to store result
Quotient in A sent to N; Start QA to store result
Yes
Exponent normal? (–63<exp<+63)
No
Request interrupt; Start QA to store result
Yes
Exponent overflow? (exp&gt;+63)
No
UNDER – UNDERFLOW (exp&lt;–127)
Clear N for zero quotient; Start QA to store result

TABLE 13.11–1 Terms Used on the KD Flow Chart for Controlling Divide and Reverse Divide Operations

AZ	A register Zero; AZ is high when the contents of register A are zero.
$\overline{\text{CD10}}$	Command Decoding bit used to distinguish between multiply and divide; CD10 is low for divide and reverse divide.
CHE ⇒ Set CED	Path enabled to effect octal exponent increment during octal normalization of the denominator. (CHE = CHange octal Exponent signal; CED = Change Exponent Delay signal that will, in this case, set CEP for enabling of ES(+1)EA path.)
DKD	DONE KD sequencer; this signal is non–existent; it is equivalent to the following: GQA, reset CZ, clear KD.
DQQ	DONE signal from the QQ sequencer.
ES6	Bit 6 in register ES; ES6 is low when an exponent underflow occurs, high for any other negative exponent range.
EUF	Exponent UnderFlow signal; EUF is high when the exponent held in register ES is in the –127 to –64 range.
EUU	Exponent Under–Underflow signal; EUU is high when the exponent held in register ES is in the –191 to –128 range.
EVF	Exponent oVerFlow signal; EVF is high when the exponent held in register ES is in the +64 to +127 range; if EVF is high, an interrupt is requested.
EVV	Exponent oVer–oVerflow signal; EVV is high when the exponent held in register ES is in the +128 to +191 range; if EVV is high, an interrupt is requested.
EXN	EXponent Normal signal; EXN is high when the exponent held in register ES is in the –63 to +63 range.
GQQ	GO signal to the QQ sequencer.
JLE	Interrupt request set if attempt has been made to divide by zero.
LJC	Signal sent by KC to start the KD and QA sequencers; LJC is equivalent to GO KD.
N41	Bit 41 in register N; N41 is high when there is a 1 in this bit position indicating binary normalization.
NN	(N) octal Normalized signal; NN is high when a 1 bit occurs in any of the 3 most significant binary positions in register N.
QAH	State of the QA sequencer; when QA has advanced to QAH, KD can continue with its own operations.
S44	Bit 44 in register S; when S44 is high, the denominator is binary normalized; (this is equivalent to the N41 signal since a left 3 shift occurred between N and S).
SZ	S register Zero signal; SZ is high when the contents of register S are zero.

FIGURE 13.11–3 The KD Flow Chart for Control of Divide and Reverse Divide Operations

KD

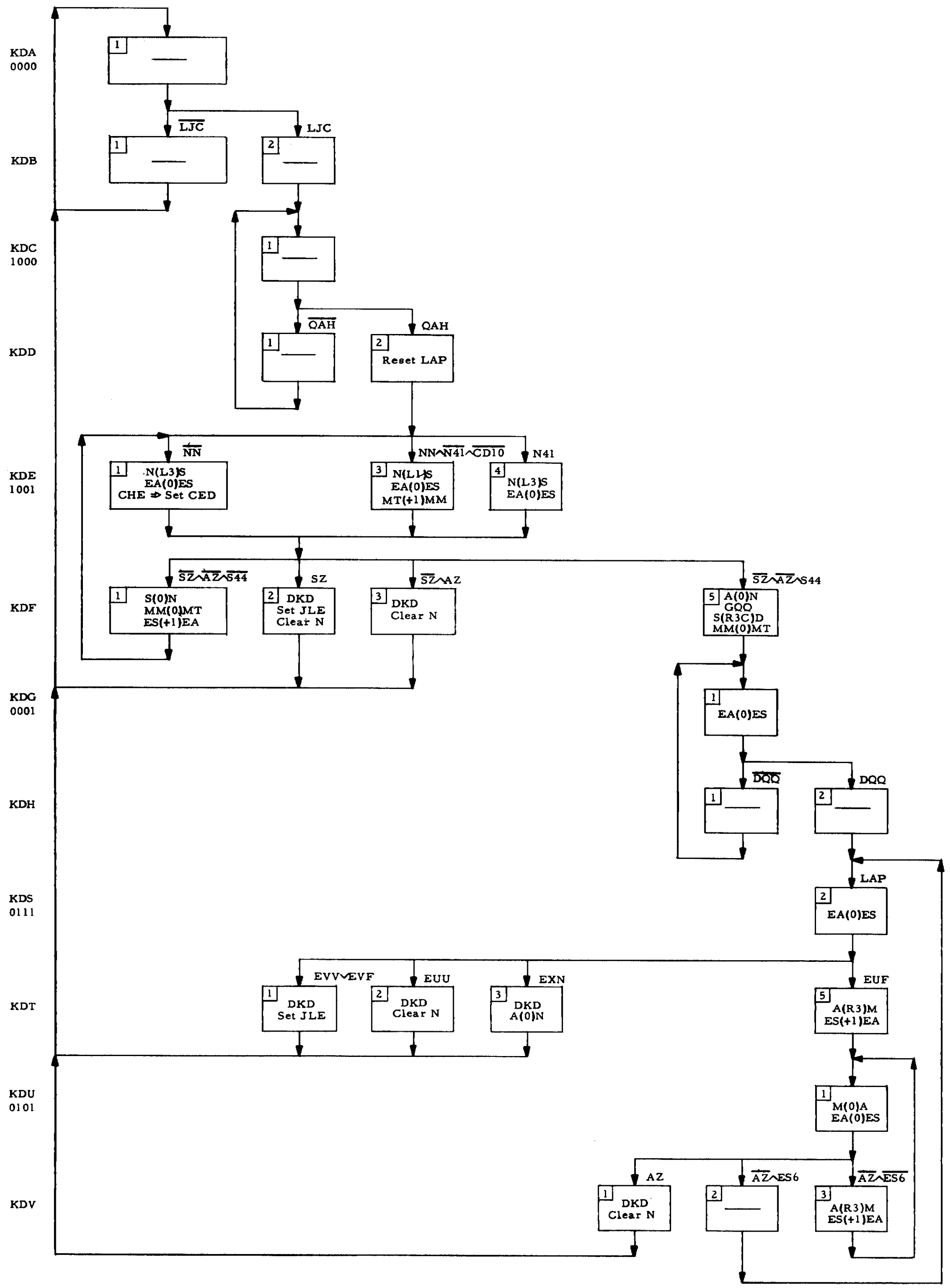
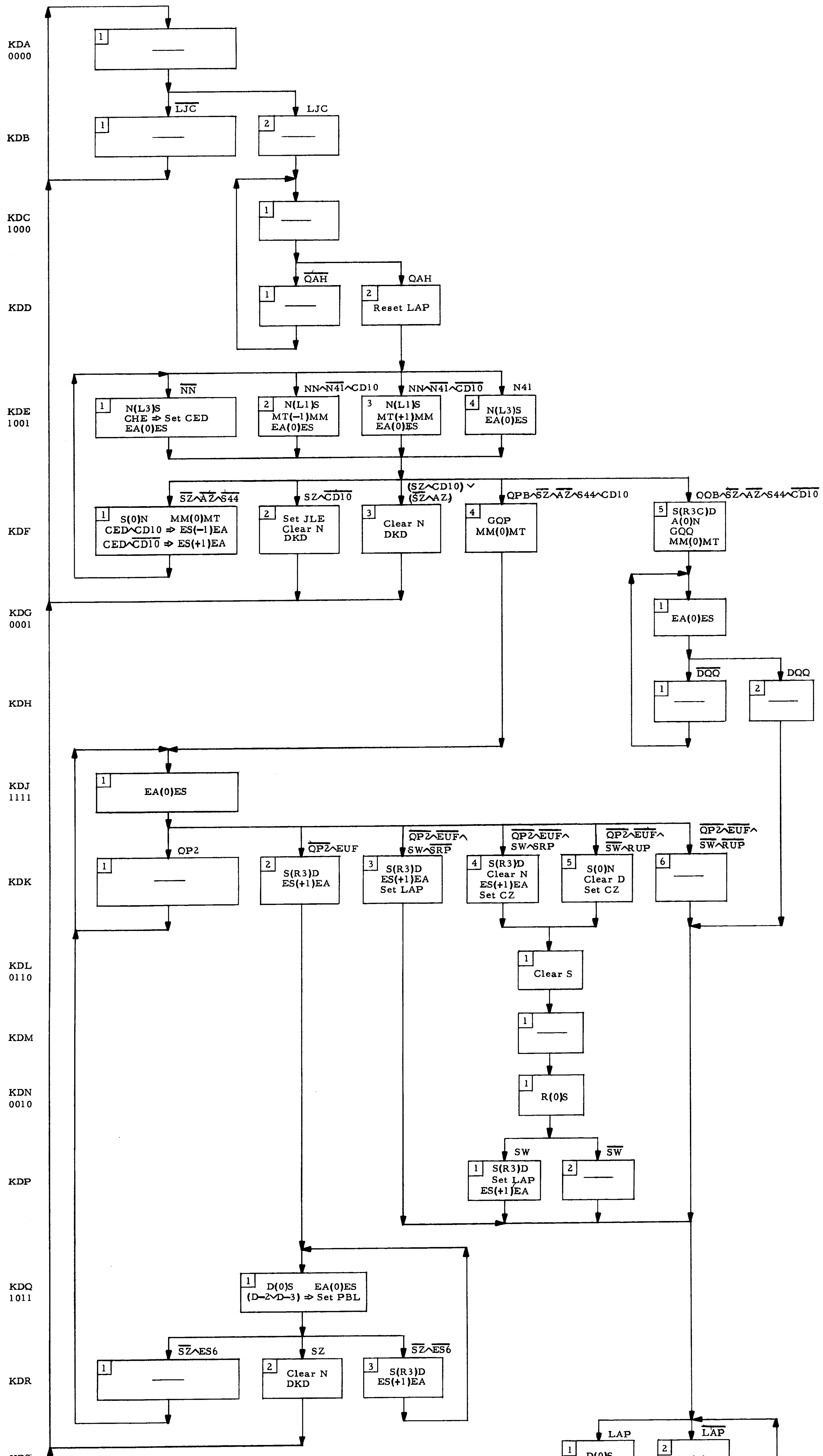
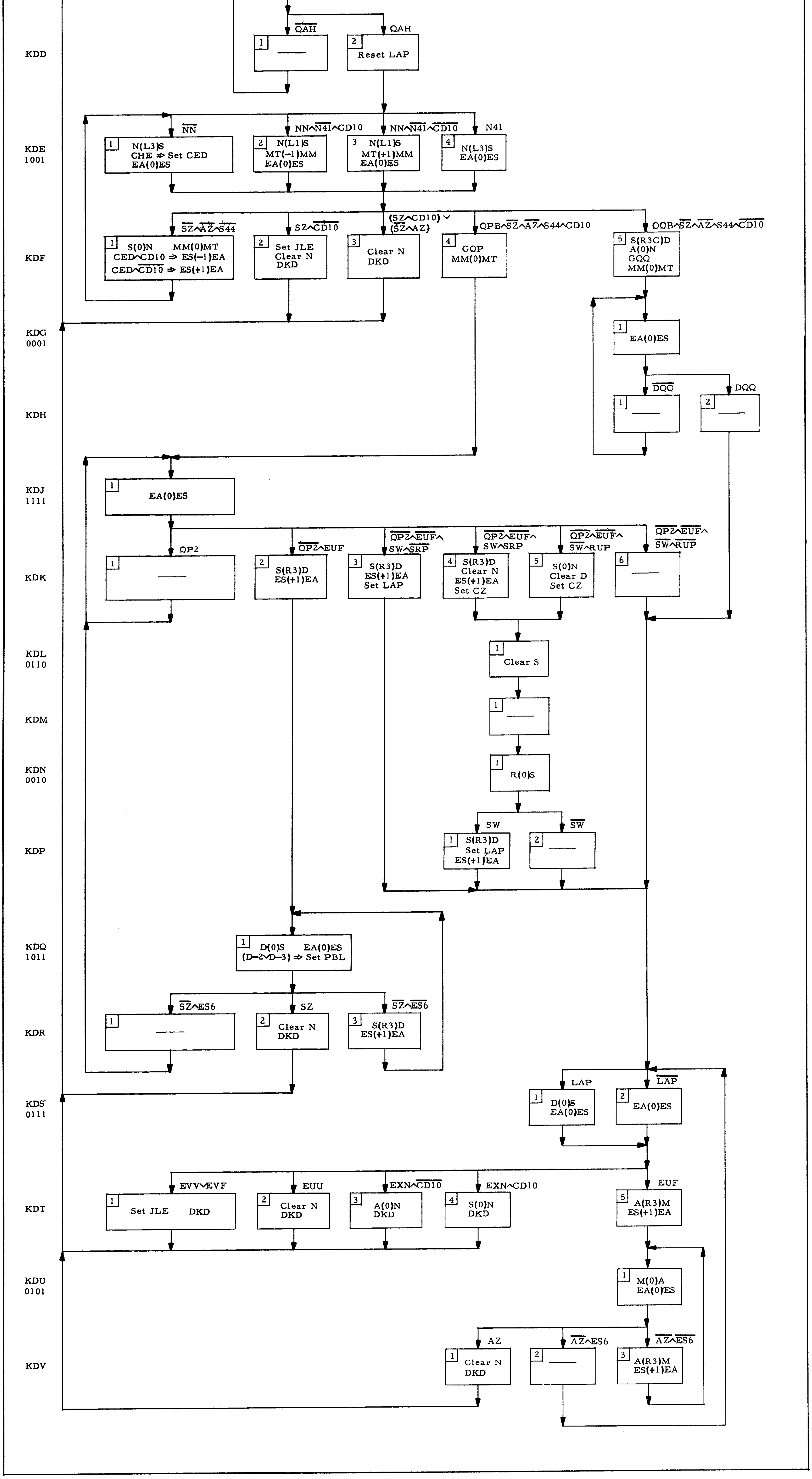


FIGURE 13.11–4 The KD Flow Chart

KD


KDA
0000
1
KDB
$\overline{LJC}$
LJC
1
2
KDC
1000
1
KDD
$\overline{QAH}$
QAH
1
2 Reset LAP
KDE
1001
$\overline{NN}$
NN∧$\overline{N41}$∧CD10
NN∧$\overline{N41}$∧$\overline{CD10}$
N41
1 N(L3)S CHE ⇒ Set CED EA(0)ES
2 N(L1)S MT(−1)MM EA(0)ES
3 N(L1)S MT(+1)MM EA(0)ES
4 N(L3)S EA(0)ES
KDF
$\overline{SZ}$∧$\overline{AZ}$∧$\overline{S44}$
SZ∧$\overline{CD10}$
(SZ∧CD10) ∨ ($\overline{SZ}$∧AZ)
QPB∧$\overline{SZ}$∧$\overline{AZ}$∧S44∧CD10
QQB∧$\overline{SZ}$∧$\overline{AZ}$∧S44∧$\overline{CD10}$
1 S(0)N MM(0)MT CED∧CD10 ⇒ ES(−1)EA CED∧$\overline{CD10}$ ⇒ ES(+1)EA
2 Set JLE Clear N DKD
3 Clear N DKD
4 GQP MM(0)MT
5 S(R3C)D A(0)N GQQ MM(0)MT
KDG
0001
1 EA(0)ES
KDH
$\overline{DQQ}$
DQQ
1
2
KDJ
1111
1 EA(0)ES
KDK
QP2
$\overline{QP2}$∧EUF
$\overline{QP2}$∧$\overline{EUF}$∧ SW∧$\overline{SRP}$
$\overline{QP2}$∧$\overline{EUF}$∧ SW∧SRP
$\overline{QP2}$∧$\overline{EUF}$∧ $\overline{SW}$∧RUP
$\overline{QP2}$∧$\overline{EUF}$∧ $\overline{SW}$∧$\overline{RUP}$
1
2 S(R3)D ES(+1)EA
3 S(R3)D ES(+1)EA Set LAP
4 S(R3)D Clear N ES(+1)EA Set CZ
5 S(0)N Clear D Set CZ
6
KDL
0110
1 Clear S
KDM
1
KDN
0010
1 R(0)S
KDP
SW
$\overline{SW}$
1 S(R3)D Set LAP ES(+1)EA
2
KDQ
1011
1 D(0)S EA(0)ES (D−2∨D−3) ⇒ Set PBL
KDR
$\overline{SZ}$∧ES6
SZ
$\overline{SZ}$∧$\overline{ES6}$
1
2 Clear N DKD
3 S(R3)D ES(+1)EA
LAP
$\overline{LAP}$
1 D(0)S
2


KDD
$\overline{QAH}$
1 ——
QAH
2 Reset LAP
KDE 1001
$\overline{NN}$
1 N(L3)S CHE ⇒ Set CED EA(0)ES
$NN \wedge \overline{N41} \wedge CD10$
2 N(L1)S MT(−1)MM EA(0)ES
$NN \wedge \overline{N41} \wedge \overline{CD10}$
3 N(L1)S MT(+1)MM EA(0)ES
N41
4 N(L3)S EA(0)ES
KDF
$\overline{SZ} \wedge \overline{AZ} \wedge \overline{S44}$
1 S(0)N MM(0)MT CED∧CD10 ⇒ ES(−1)EA CED∧$\overline{CD10}$ ⇒ ES(+1)EA
$SZ \wedge \overline{CD10}$
2 Set JLE Clear N DKD
$(SZ \wedge CD10) \vee (\overline{SZ} \wedge AZ)$
3 Clear N DKD
$QPB \wedge \overline{SZ} \wedge \overline{AZ} \wedge S44 \wedge CD10$
4 GQP MM(0)MT
$QQB \wedge \overline{SZ} \wedge \overline{AZ} \wedge S44 \wedge \overline{CD10}$
5 S(R3C)D A(0)N GQQ MM(0)MT
KDG 0001
1 EA(0)ES
KDH
$\overline{DQQ}$
1 ——
DQQ
2 ——
KDJ 1111
1 EA(0)ES
KDK
QP2
1 ——
$\overline{QP2} \wedge EUF$
2 S(R3)D ES(+1)EA
$\overline{QP2} \wedge \overline{EUF} \wedge SW \wedge \overline{SRP}$
3 S(R3)D ES(+1)EA Set LAP
$\overline{QP2} \wedge \overline{EUF} \wedge SW \wedge SRP$
4 S(R3)D Clear N ES(+1)EA Set CZ
$\overline{QP2} \wedge \overline{EUF} \wedge \overline{SW} \wedge RUP$
5 S(0)N Clear D Set CZ
$\overline{QP2} \wedge \overline{EUF} \wedge \overline{SW} \wedge \overline{RUP}$
6 ——
KDL 0110
1 Clear S
KDM
1 ——
KDN 0010
1 R(0)S
KDP
SW
1 S(R3)D Set LAP ES(+1)EA
$\overline{SW}$
2 ——
KDQ 1011
1 D(0)S EA(0)ES (D−2∨D−3) ⇒ Set PBL
KDR
$\overline{SZ} \wedge ES6$
1 ——
SZ
2 Clear N DKD
$\overline{SZ} \wedge \overline{ES6}$
3 S(R3)D ES(+1)EA
KDS 0111
LAP
1 D(0)S EA(0)ES
$\overline{LAP}$
2 EA(0)ES
KDT
EVV∨EVF
1 Set JLE DKD
EUU
2 Clear N DKD
$EXN \wedge \overline{CD10}$
3 A(0)N DKD
$EXN \wedge CD10$
4 S(0)N DKD
EUF
5 A(R3)M ES(+1)EA
KDU 0101
1 M(0)A EA(0)ES
KDV
AZ
1 Clear N DKD
$\overline{AZ} \wedge ES6$
2 ——
$\overline{AZ} \wedge \overline{ES6}$
3 A(R3)M ES(+1)EA

SECTION 13.12 – QUOTIENT FORMATION BY THE QQ SEQUENCER

In binary division the quotient is formed by means of determination of the orders at which $r > d$ (where r is the remainder, d the denominator) and supplying 1 bits in the quotient at those orders. r is compared to d by means of subtractions of d from r. The result of each subtraction is the new remainder that will be used in the subsequent comparison of d to r. If the new remainder is positive, the quotient receives a 1 bit.

In Section 13.8 it was demonstrated that the operands must be properly positioned before each subtraction. In the G–20 this positioning is accomplished through left shifts of the remainder along with appropriate changes in the current quotient exponent value. At GO QQ these operands are positioned as follows:

1) The denominator has been binary normalized, then complemented, and stored in register D; it remains there, unchanged, throughout the operation of QQ;

3) The numerator, (the first remainder value) is in register N; it will be shifted left until it is binary normalized to position it correctly for the first subtraction of d from r. The new remainder is gated from the Adder to S, then sent from S to N. Each new remainder will be binary normalized before a subtraction takes place. (This happens because the quotient bits that would result from subtractions occurring before the remainder become binary normalized are entirely predictable and, therefore, these iterations can be bypassed. Handling of bit determination during shifting is discussed later in this section.)

The quotient is formed between the A and M registers. These are cleared by QQ for the starting quotient value of zero: (S is cleared, then S(R3)M is enabled followed by M(0)A or M(L2)A depending upon whether or not the numerator is already binary normalized). Quotient bits are sent to the least significant bit position and the quotient is shifted left in step with the remainder. These quotient bits are 1's for each successful subtraction, i.e., wherever r > d, 0's for non–successful subtractions. The division process continues until the quotient is octal normalized and has a non–fractional exponent (the modulo–three count is equal to zero). Thus, when QQ is started, the current quotient exponent is held in the octal exponent counter and the modulo–three counter while, at DONE QQ, the current quotient exponent is held in the octal exponent counter.

The division algorithm is speeded up through use of the principal of non–restorative divide. The familiar form of division employs restorative divide. This means that, if a subtraction of d from r results in a negative remainder, i.e., if d > r, the new remainder is added back into the denominator to restore the former remainder value before a subtraction occurs at the next order. The objection to this procedure is that it involves a great many add cycles and, thus, a good deal of time.

In non–restorative divide the remainder is allowed to become negative. This condition is taken into account at the next subtraction where a minus multiple of the denominator is subtracted from the negative remainder (or, since – (– denominator) = + denominator, the denominator is added to the negative remainder). The negative quotient value at that order is added to the partial quotient. Thus, the division can continue without time being taken to restore the remainder to a positive value. Consider this principle as applied to a decimal example: 2822/12. Initially the remainder and denominator are assumed to be positive. A multiple of the denominator is subtracted from the remainder for the first quotient value; the operation performed, therefore, will be a differencing. A too–large multiple of the denominator is chosen for this first subtraction to show how a negative remainder is handled.

$$300 * 12 = 3600 \qquad \begin{array}{r} +2822 \\ \underline{-(+3600)} \\ -778 \end{array} = \text{new remainder} \qquad \text{Partial quotient} = +300$$

To correct for this condition, the next iteration will call for subtracting a minus multiple of the denominator or, since – (– denominator) = + denominator, a multiple of the denominator will be added to the negative remainder. This, again is a differencing operation.

$$-70 * 12 = -840 \qquad \begin{array}{r} -778 \\ \underline{-(-840)} \\ +62 \end{array} = \text{new remainder} \qquad \begin{array}{r} +300 \\ \underline{+(-\ 70)} \\ +230 \end{array} = \text{partial quotient}$$

The new remainder is positive. Thus, the next iteration calls for a subtraction.

$$+5 * 12 = 60 \qquad \begin{array}{r} +62 \\ \underline{-(+60)} \\ +2 \end{array} = \text{new remainder} \qquad \begin{array}{r} +230 \\ \underline{+(+\ \ 5)} \\ +235 \end{array} = \text{partial quotient}$$

The two possible cases are demonstrated in this example:

1) positive remainder value ⇒ subtract positive denominator;
2) negative remainder value ⇒ subtract negative denominator; (add denominator).

Thus, the subtractions called for during division always involve differencing operations. To obtain a difference through use of the Adder, one of the operands must be complemented. Since the normal case is that in which $r > d$, the denominator is complemented and stored in register D. This complemented value is always one of the Adder inputs.

Differencing as performed by QQ is almost identical to that carried by QS. The distinctions between the two cases are pointed out here to avoid confusion. In QS, (N) are complemented. This is an arbitrary choice of operand since it doesn't matter which is larger or which is complemented so long as the output is handled properly in either case. The normal case in QS is taken to be that in which /D/>/N/. Generation of a CY signal occurs when the uncomplemented operand, in this case (D), has the larger absolute value; this indicates that the output is in normal form. (CZ is set before each differencing operation in QS to cause a carry in to the zero order; this corrects the normal output which would otherwise be low by 1.) In divide, the usual case is that in which (N) (the remainder or, for the first iteration, the numerator) is larger than (D) (the denominator). For each such case the quotient receives a 1 bit. A differencing operation performed by QQ when the remainder is positive (remainder sign is recorded in the Remainder Sign flip–flop, RS, which is low for positive) and which results in a CY signal indicates /N/ is larger. It can be stated that for $\overline{\text{RS}} \wedge \text{CY}$:

1) /N/>/D/ and $r \geq d$;

2) the new remainder is positive and in normal form; (CZ is set throughout the operation of QQ);

3) the subtraction was successful; the quotient receives a 1 bit in the appropriate order.

In QS, generation of $\overline{\text{CY}}$ indicates that the complemented number has the larger absolute value, and therefore, the Adder output is in complement form. QS, on detection of $\overline{\text{CY}}$, resets CZ and re–enters the add cycle to avoid a carry in. The result is then in 1's complement form. This is handled differently in QQ where CZ remains set for all cases. Thus, the complement that is formed when $\overline{CY}$ is high is assumed to be 1 greater than the 1's complement of the result, or in 2's complement form. The complement paths in the G–20 form the 1's complement so that complementation of the remainder value yields a converted remainder that is in normal form, but is low by 1. Thus, for the case $\overline{RS} \wedge \overline{CY}$:

1) /D/>/N/ and d > r;

2) the remainder is negative and in 2's complement form; after conversion (1's complementing) it will be low by 1;

3) the subtraction was not successful; the quotient receives a 0 bit in the appropriate order.

In QS a single result is required. When differencing, determination of the larger absolute value and certain sign manipulations complete the operation. In QQ a cumulative remainder is formed. Thus, the occurrence of a negative remainder and its conversion to the form "normal –1" will be followed by another subtraction as soon as the remainder has been shifted to the binary normalized position. As has been demonstrated, the quantity formed is $[-r-(-d)]$ or $(-r+d)$. If at this iteration d is still larger than r, the new remainder will again be negative and the quotient will again receive a 0 bit. If r is greater than d, the remainder will become positive. For this case also, the CY signal is used to determine which operand is larger. However, the subtraction between two negative values changes the interpretation of the output. It is obvious that, for negative values, the larger one is that with the smaller absolute value. (Thus, of the numbers –7 and –3, the latter is the larger of the two.) It has been shown that for QQ, generation of a CY signal indicates /N/≥/D/. Since for negative values the meaning attached to the larger absolute value is reversed, CY indicates d > r for the $[-r-(-d)]$ case whereas it indicates r ≥ d for the [+r − (+d)] case. It is important to notice that this consideration does not affect the interpretation of the form of the output. This is always dependent upon which operand has the larger absolute value. Thus, CY indicates that the absolute value of the uncomplemented operand is greater than or equal to that of the other operand and that the result is therefore in normal form. Conversely, $\overline{CY}$ always indicates /D/ is larger and, therefore, that the output is in complement form. Given this information, it can be stated that for RS $\wedge$ CY:

1) /N/≥/D/ and, since both r and d are negative, d ≥ r;

2) the remainder is still negative; the uncomplemented operand has the larger absolute value so that the result is in normal form; CZ is set for the necessary carry in; however, the result is still low by 1 due to the fact that no correction was made for CZ when the remainder became negative; this error is propagated until the remainder again becomes positive;

3) the subtraction was not successful since d > r; quotient bit = 0;

and for RS $\wedge$ $\overline{CY}$:

1) /D/>/N/ and, since both r and d are negative, r > d;

2) the remainder has become positive; the complemented operand has the larger absolute value so that the result is in complement form; CZ is still set, thus correcting for the fact that the remainder has been low by 1 while negative;

3) the subtraction was successful; therefore, the quotient bit is 1.

For each subtraction, the remainder and quotient are shifted left 1 bit and the current quotient exponent is decremented by 1/3. As has been stated before, the quotient receives a 1 in the least significant bit position if r ≥ d, a 0 if r < d. The remainder receives a 0 if the remainder value is positive, a 1 if it is negative. This avoids multiplying the error that is present while the remainder is negative (the remainder is low by 1 because CZ was not reset on the unsuccessful subtraction). If zeros were inserted on the shifts, the remainder would become lower by a factor of 2 with each shift. The insertion of 1 bit keeps it low by exactly 1 no matter how many shifts occur before it goes positive again.

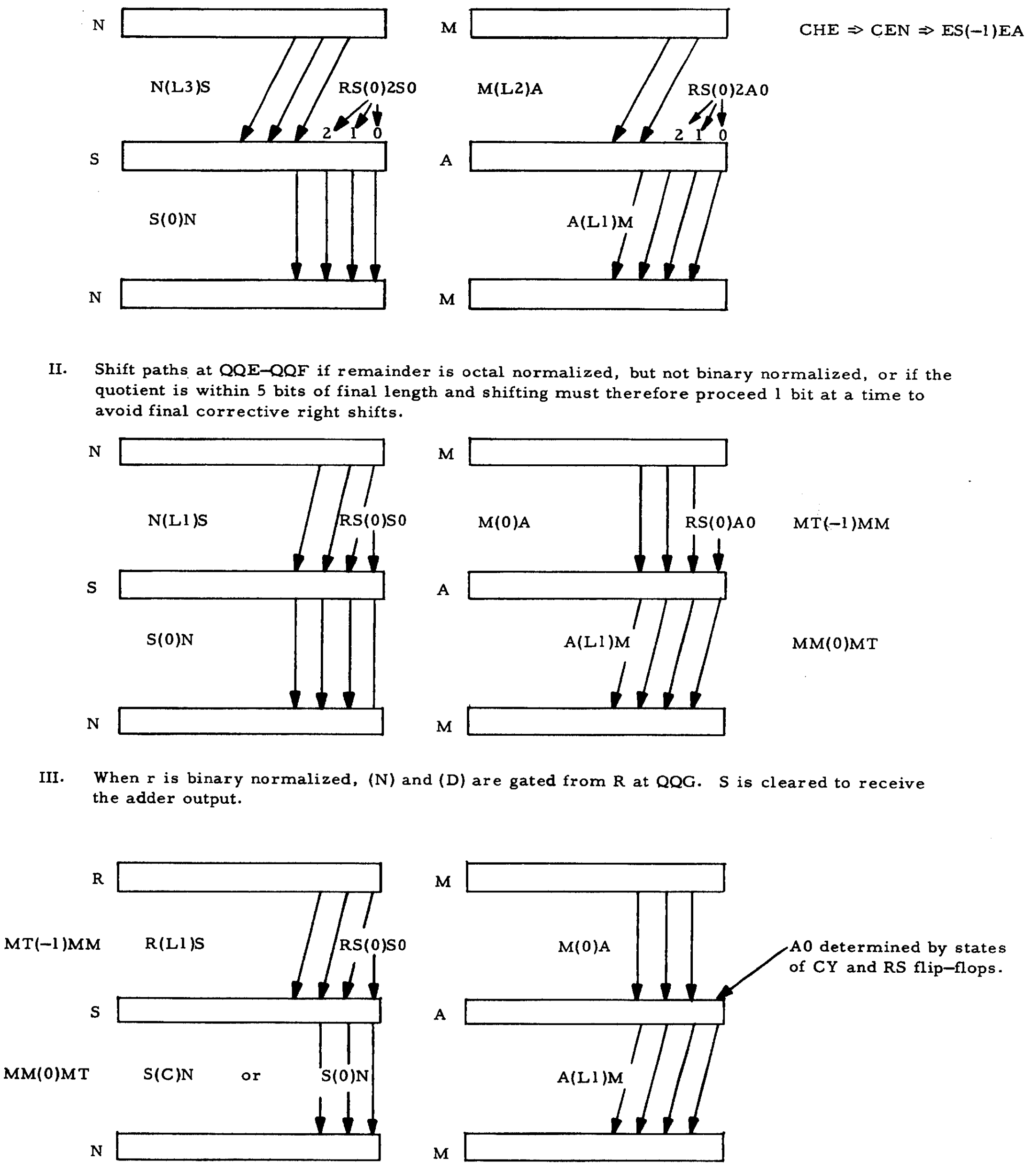
When the remainder is not binary normalized, the quotient and the remainder are shifted to the left 3 bits at a time until octal normalization of the remainder occurs, then 1 bit at a time until it is binary normalized. The exponent is decremented appropriately. On these shifts, both the quotient and the remainder receive 0's in the least significant bits if the remainder is positive, 1's if it is negative. For the remainder, the reason is that mentioned above. For the quotient value, the choice of bits results from the fact that the result of any subtraction attempted before binary normalization of the remainder occurs is predictable. Recall that the denominator is binary normalized before being complemented, and stored in D. Thus, when the remainder is not binary normalized, $/D/>/N/$ so that all subtractions performed under these conditions would result in signal $\overline{CY}$. Thus, it is possible to determine the correct quotient bits for these orders without performing any subtraction operation before normalization of the remainder occurs. For $\overline{RS} \wedge \overline{CY}$, the quotient bit is 0; for $RS \wedge \overline{CY}$, the quotient receives a 1. This information, along with other relevant points made in this section, is summarized in Table 13.12–1.

Three shifting sequences are used in QQ. At GO QQ, the denominator has been binary normalized, complemented, and stored in D. It remains there, unchanged, throughout the operation. A differencing operation will occur only when the remainder is also binary normalized. The first shift shown in Figure 13.12–1 is that used to octal normalize the remainder. Following octal normalization (indicated by signal NN), the remainder is binary normalized. This constitutes the second shift shown in the figure. The differencing cycle itself involves a 1–bit shift to the left. This is shown as the third shift sequence. Following the differencing cycle, the remainder is again binary normalized. Figure 13.12–2 is the QQ algorithm, Figure 13.12–3, the flow chart.

TABLE 13.12–1 Actions Taken on the Basis of the Remainder Sign and Carry Signal During the Operation of QQ

	REMAINDER NOT BINARY NORMALIZED		REMAINDER BINARY NORMALIZED				
	Shifts	Bit Determination During Shifting	Action	Interpretation of CY signal: CY ⇒ /N/⩾/D/ or /r/⩾/d/			Remainder Sign
				Form of result	Size of Operands	Bit Inserted on Left 1 Shift	
$\overline{RS} \wedge CY$	Shift remainder and quotient left until remainder binary normalized; decrement current quotient exponent in step;	$\overline{RS}$ ⇒ 0's in least significant bit positions of quotient and remainder	Subtract positive denominator complement from positive remainder [+r–(+dc)]	CY ⇒ normal	r ⩾ d	quotient = 1 remainder = 0	New remainder still positive
$\overline{RS} \wedge \overline{CY}$			Shift left 1, decrement exponent by 1/3	$\overline{CY}$ ⇒ complement; (2's complement since CZ set)	d > r	quotient = 0 remainder = 0	New remainder has become negative; therefore set RS
RS ∧ CY		RS ⇒ 1's in least significant bit positions of quotient and remainder	Subtract negative denominator complement from negative remainder [–r–(–dc)] = (–r +dc) Shift left 1, decrement exponent by 1/3.	CY ⇒ normal; (low by 1 after conversion from 2's complement)	d > r (negative values being compared)	quotient = 0 remainder = 1	New remainder still negative
$RS \wedge \overline{CY}$				$\overline{CY}$ ⇒ complement; (1's complement)	r ⩾ d (negative values being compared)	quotient = 1 remainder = 1	New remainder has become positive; therefore reset RS

FIGURE 13.12–1 Shift Paths Used in QQ



I. Shift paths at QQE–QQF if remainder is not octal normalized and the quotient is not within 5 bits of final length (14 octals). This operation is repeated until either NN signals octal normalization or CPQ signals that the quotient is within 5 bits of final length.

II. Shift paths at QQE–QQF if remainder is octal normalized, but not binary normalized, or if the quotient is within 5 bits of final length and shifting must therefore proceed 1 bit at a time to avoid final corrective right shifts.

III. When r is binary normalized, (N) and (D) are gated from R at QQG. S is cleared to receive the adder output.

FIGURE 13.12–2 Algorithm for the Formation of 14 Octal Quotient by QQ

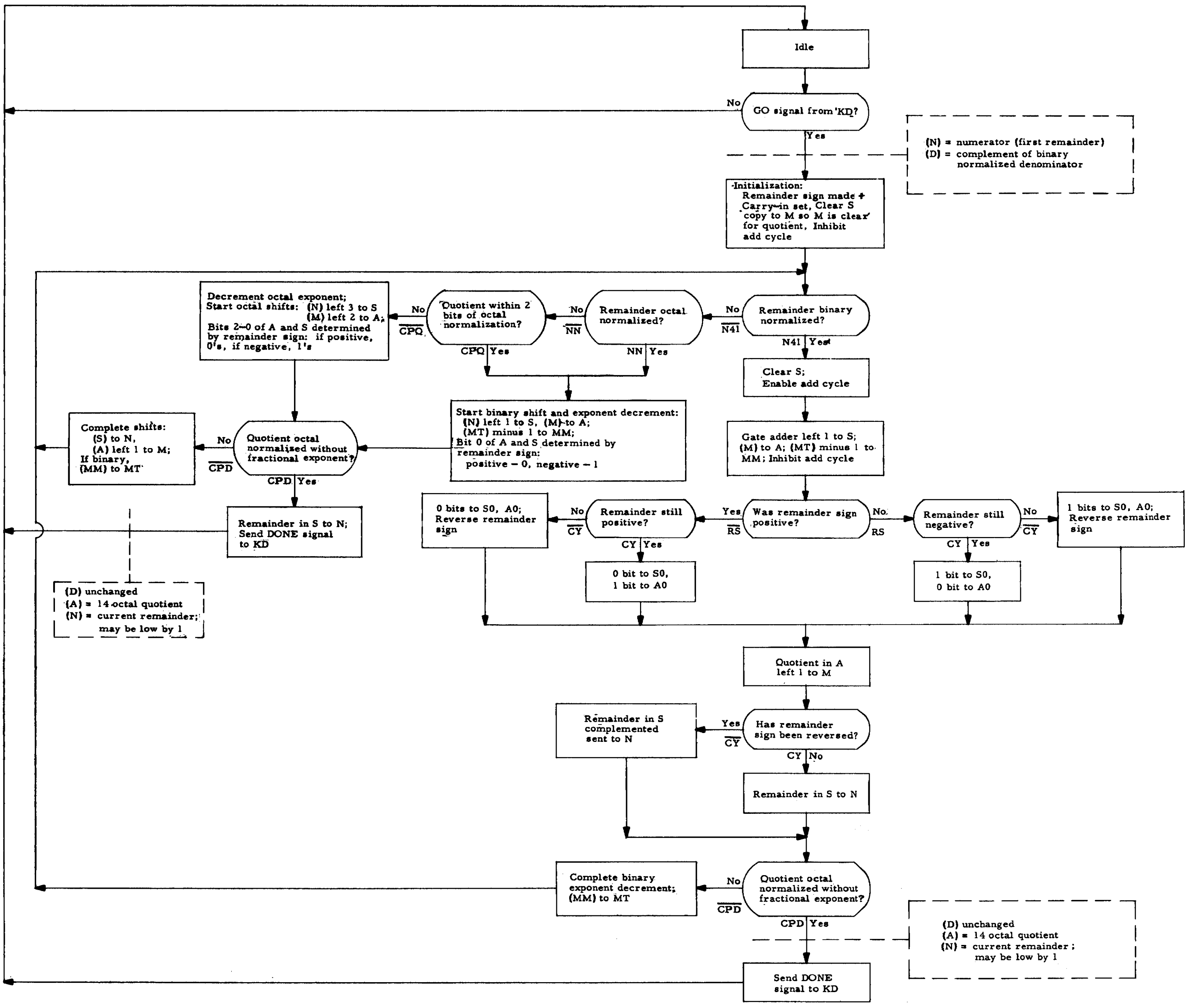
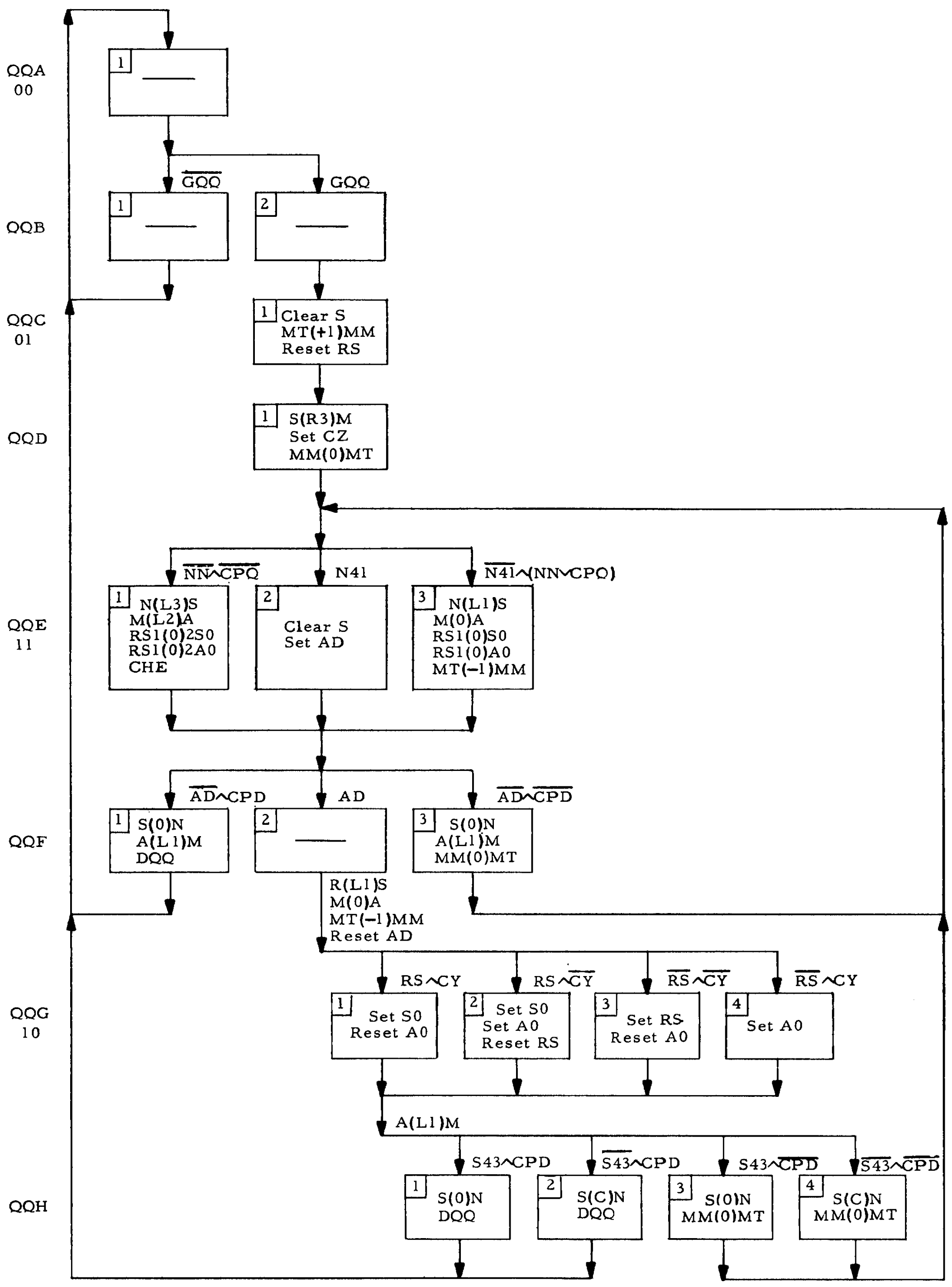

Idle
No
GO signal from 'KD?
Yes
(N) = numerator (first remainder)
(D) = complement of binary normalized denominator
Initialization: Remainder sign made + Carry-in set, Clear S copy to M so M is clear for quotient, Inhibit add cycle
Remainder binary normalized?
No
$\overline{\text{N41}}$
N41 Yes
Remainder octal normalized?
No
$\overline{\text{NN}}$
NN Yes
Quotient within 2 bits of octal normalization?
No
$\overline{\text{CPQ}}$
CPQ Yes
Decrement octal exponent; Start octal shifts: (N) left 3 to S (M) left 2 to A; Bits 2–0 of A and S determined by remainder sign: if positive, 0's, if negative, 1's
Start binary shift and exponent decrement: (N) left 1 to S, (M) to A; (MT) minus 1 to MM; Bit 0 of A and S determined by remainder sign: positive – 0, negative – 1
Quotient octal normalized without fractional exponent?
No
$\overline{\text{CPD}}$
CPD Yes
Complete shifts: (S) to N, (A) left 1 to M; If binary, (MM) to MT
Remainder in S to N; Send DONE signal to KD
(D) unchanged
(A) = 14 octal quotient
(N) = current remainder; may be low by 1
Clear S; Enable add cycle
Gate adder left 1 to S; (M) to A; (MT) minus 1 to MM; Inhibit add cycle
Was remainder sign positive?
Yes
$\overline{\text{RS}}$
No
RS
Remainder still positive?
No
$\overline{\text{CY}}$
CY Yes
0 bits to S0, A0; Reverse remainder sign
0 bit to S0, 1 bit to A0
Remainder still negative?
No
$\overline{\text{CY}}$
CY Yes
1 bit to S0, 0 bit to A0
1 bits to S0, A0; Reverse remainder sign
Quotient in A left 1 to M
Has remainder sign been reversed?
Yes
$\overline{\text{CY}}$
CY No
Remainder in S complemented sent to N
Remainder in S to N
Quotient octal normalized without fractional exponent?
No
$\overline{\text{CPD}}$
CPD Yes
Complete binary exponent decrement; (MM) to MT
(D) unchanged
(A) = 14 octal quotient
(N) = current remainder ; may be low by 1
Send DONE signal to KD

TABLE 13.12–2 Terms Used on the QQ Flow Chart

AD2	Add Delay flip–flop; AD is reset to avoid entry into the Adder section of QQ, set to gate the result from the Adder.
AN	(A) octal Normalized; AN is high when any significant bits occur in the most significant octal position of register A.
CHE	CHange octal Exponent signal.
CPD	Quotient Cleanup Phase Done signal; CPD is high if AN is high and the current exponent value is non–fractional, i.e., the modulo–three counter contains zeros; when CPD is high, QQ exits.
CPQ	Cleanup Phase, Quotient signal; $CPQ = M41 \vee M40 \vee M39 \vee M38 \vee M37$; CPQ is used to prevent left–3 shifts from occurring if a 1 bit is already stored in any of the five high–order positions in M; this is desirable since final octal justification of the quotient might require right shifts if left–3 shifts were allowed at this stage.
CY	CarrY out of Adder flip–flop; during the operation of QQ, the occurrence of $\overline{CY}$ causes the state of the RS flip–flop to be reversed.
CZ	Carry into Zero order of the Adder; CZ remains set throughout the operation of QQ.
DQQ	DONE signal from the QQ sequencer.
EA(0)ES	Path enabled on all C1's while QQ is in operation since KD is idling at KDG–KDH where this enabling occurs. This updates the octal exponent counter on each clock.
ES(–1)EA	Path enabled to decrement octal exponent; enabling occurs on C2's if $CHE \vee (QQ3 \wedge MT(-1)MM \wedge \overline{MT1} \wedge \overline{MT0})$.
ES(+1)EA	Path enabled at QQC when exponent is incremented by 1/3 if the current count in the modulo–three counter is 2/3; thus, if MT = 10, MT(+1)MM causes enabling of this path $(C2 \wedge QQC \wedge MT(+1)MM \wedge MT1 \wedge \overline{MT0})$.
GQQ	GO signal to QQ sequencer from KD.
N41	Bit 41 in register N; N41 is high if the 42nd bit in the remainder is a 1, and, thus, indicates that the remainder is binary normalized.
NN	(N) octal Normalized; NN is high when any significant bits are stored in the most significant octal position in register N.
R(L1)S	Path enabled to gate the output from the Adder to register S.
RS	Remainder Sign flip–flop; RS high indicates a negative sign, low, a positive sign.
RS(0)2S0	Path enabled to copy the condition of the RS flip–flop into the 3 least significant bits in register S.
S43	Bit 43 in register S; S43 is high following the gating of the result from the Adder if CY is high and enabled by CYE and if the output is gated by means of the R(L1)S path.

FIGURE 13.12–3 The QQ Flow Chart

QQ



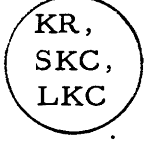
CHAPTER 14

APPENDIX I

REPEAT MODE BLOCK DIAGRAM

A block diagram of sequencer operation in the repeat mode is included here for reference purposes. This diagram has been used in the Customer Engineering Central Processor courses and has proven useful to those who are familiar with the operation of the individual sequencers but who do not fully grasp the relationship of these sequencers to one another. This diagram shows the interactions particularly well since it is organized on the basis of the signals used for communication between sequencers rather than on the basis of the functions being performed by each sequencer. Thus, anyone familiar with the basic concepts involved can determine from this diagram where the signals that relate to these functions are generated and thereby accustom himself to the overall picture. In addition, the inclusion of all possible repeat operations (those processed by KL as well as those processed by KA) on the same chart facilitates comparison between the various operations. The following conventions are observed on the drawing:

1) circles indicate idle loops. Thus,



indicates that the sequencer is idling and that it will advance from this idle loop on receipt of a KR, SKC or LKC signal.

2) solid lines indicate paths through the states of the indicated sequencer;

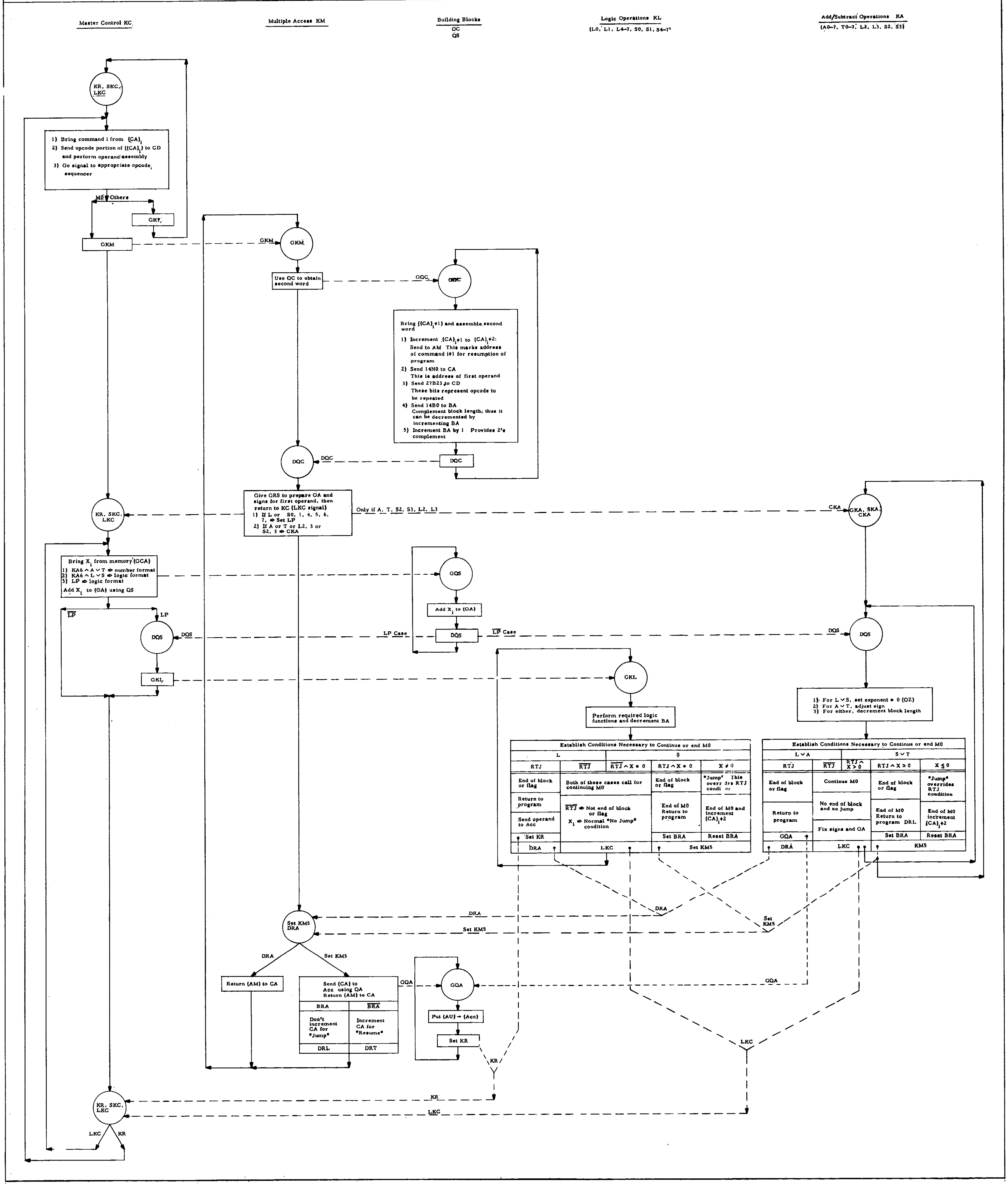
3) columns are sequencer oriented. Thus, information in column 1 pertains to Master Control, column 2 to the KM sequencer, column 3 the building block sequencers QC and QS, column 4 the repeat mode of KL and column 5 the repeat mode of KA.

4) rows are time oriented with the first operation being indicated in the upper left hand corner. Thus, the operation is initiated by KC, KC starts KM in the second row, KM starts QC in the succeeding row, and so forth.

5) dotted lines represent the various forms of communication between sequencers, i.e., the origins and destinations of signals.

For a detailed description of the repeat mode, see Sections 13.2 and 13.3.

FIGURE 14–1 Block Diagram of Repeat Mode


Master Control KC
Multiple Access KM
Building Blocks
QC
QS
Logic Operations KL
(L0, L1, L4–7, S0, S1, S4–7'
Add/Subtract Operations KA
(A0–7, T0–7, L2, L3, S2, S3)
KR, SKC, LKC
1) Bring command i from (CA)ᵢ
2) Send opcode portion of ((CA)ᵢ) to CD and perform operand assembly
3) Go signal to appropriate opcode sequencer
M0
Others
GK?
GKM
GKM
GKM
Use QC to obtain second word
GQC
GQC
Bring ((CA)ᵢ+1) and assemble second word
1) Increment (CA)ᵢ+1 to (CA)ᵢ+2: Send to AM This marks address of command i+1 for resumption of program
2) Send 14N0 to CA This is address of first operand
3) Send 27B23 to CD These bits represent opcode to be repeated
4) Send 14B0 to BA Complement block length; thus it can be decremented by incrementing BA
5) Increment BA by 1 Provides 2's complement
DQC
DQC
DQC
Give GRS to prepare OA and signs for first operand, then return to KC (LKC signal)
1) If L or S0, 1, 4, 5, 6, 7, ⇒ Set LP
2) If A or T or L2, 3 or S2, 3 ⇒ CKA
Only if A, T, S2, S3, L2, L3
CKA
GKA, SKA, CKA
KR, SKC, LKC
Bring Xᵢ from memory (GCA)
1) KA6 ∧ A ∨ T ⇒ number format
2) KA6 ∧ L ∨ S ⇒ logic format
3) LP ⇒ logic format
Add Xᵢ to (OA) using QS
GQS
Add Xᵢ to (OA)
LP
LP
DQS
DQS
LP Case
DQS
LP Case
DQS
DQS
GKL
GKL
Perform required logic functions and decrement BA
1) For L ∨ S, set exponent = 0 (QZ)
2) For A ∨ T, adjust sign
3) For either, decrement block length
Establish Conditions Necessary to Continue or end M0
L
S
RTJ
RTJ
RTJ ∧ X = 0
RTJ ∧ X = 0
X ≠ 0
End of block or flag
Return to program
Send operand to Acc
Set KR
DRA
Both of these cases call for continuing M0
RTJ ⇒ Not end of block or flag
Xᵢ ⇒ Normal "No Jump" condition
LKC
End of block or flag
End of M0 Return to program
Set BRA
"Jump" This overrides RTJ condition
End of M0 and increment (CA)ᵢ+2
Reset BRA
Set KM5
Establish Conditions Necessary to Continue or end M0
L ∨ A
S ∨ T
RTJ
RTJ
RTJ ∧ X > 0
RTJ ∧ X > 0
X ≤ 0
End of block or flag
Return to program
GQA
DRA
Continue M0
No end of block and no jump
Fix signs and OA
LKC
End of block or flag
End of M0 Return to program DRL
Set BRA
"Jump" overrides RTJ condition
End of M0 increment (CA)ᵢ+2
Reset BRA
KM5
DRA
Set KM5
DRA
Set KM5
Set KM5 DRA
DRA
Set KM5
Return (AM) to CA
Send (CA) to Acc using QA Return (AM) to CA
BRA
BRA
Don't increment CA for "Jump"
Increment CA for "Resume"
DRL
DRT
GQA
GQA
GQA
Put (AU) → (Acc)
Set KR
KR
LKC
KR
LKC
KR, SKC, LKC
LKC
KR

APPENDIX II

CD DECODING FOR GROUPS OF COMMANDS

Group Designation	Selective Bits in Register CD	Designated Commands
CDA	$CD11 \wedge CD10 \wedge \overline{CD8} \wedge CD7$	P2 ∨ P4 ∨ R3
CDB	$[\overline{CD9} \wedge CD7] \vee [\overline{CD6} \wedge (CD9 \vee CD8 \vee CD7)]$	B0–7 ∨ R0–3
CDC	$CD9 \wedge [CD12 \vee \overline{CD11} \vee \overline{CD6}]$	B0–1 ∨ B4–5 ∨ L0–1 ∨ L4–7 ∨ M0–1 ∨ P0–3 ∨ R0–3 ∨ S0–1 ∨ S4–7 ∨ X0–5
CDD	$[\overline{CD12} \wedge CD11 \wedge (\overline{CD10} \vee CD8 \vee \overline{CD7})] \vee [\overline{CD9} \wedge \overline{CD6}]$	A2–3 ∨ B2–3 ∨ B6–7 ∨ D0–2 ∨ L2–3 ∨ N0–7 ∨ R0–2 ∨ S2–3 ∨ T2–3
CDE	$CD12 \wedge CD8 \wedge (\overline{CD11} \vee \overline{CD10})$	A4–6 ∨ L4–6 ∨ X2 ∨ X4–5
CDF	$\overline{CD12} \wedge \overline{CD11} \wedge CD7 \wedge \overline{CD6}$	B0–7
CDG	$\overline{\overline{CD12} \wedge CD7 \wedge \overline{CD6}}$ $\overline{CDG}$ = CDB = CDK = B0–7 ∨ R0–3	$\overline{B0\text{–}7 \vee R0\text{–}3}$
CDH	$CD11 \wedge \overline{CD9}$	A2–2 ∨ A6–7 ∨ N2–3 ∨ N6–7 ∨ T2–3 ∨ T6–7
CDJ		A0–7 ∨ D0–2 ∨ L2–3 ∨ N0–7 ∨ S2–3 ∨ T0–7
CDK	$\overline{CD12} \wedge CD7 \wedge CD6 \wedge (\overline{CD11} \vee CD9)$	B0–7 ∨ R0–3
CDL	$CD7 \wedge [(CD11 \wedge CD10) \vee \overline{CD12} \vee \overline{CD8}]$	B0–7 ∨ D0–2 ∨ M0–1 ∨ P0–4 ∨ R0–3 ∨ X0–1 ∨ X3

Group Designation	Selective Bits in Register CD	Designated Commands
CDN	$\overline{CD12} \wedge \overline{CD11} \wedge \overline{CD9} \wedge \overline{CD8} \wedge \overline{CD7} \wedge \overline{CD6}$	N0–1
CDQ	$\overline{CD9} \wedge \overline{CD8} \wedge \overline{CD7} \wedge \overline{CD6}$	N0–7
CDR	$\overline{CD12} \wedge CD11 \wedge \overline{CD10} \wedge CD9 \wedge \overline{CD8} \wedge CD7 \wedge \overline{CD6}$	R2
CDS	$\overline{CD12} \wedge \overline{CD11} \wedge CD10$	N1 ∨ T1 ∨ A1 ∨ L1 ∨ S1 ∨ B1 ∨ M1 ∨ X1 ∨ B5 ∨ B7 ∨ B3
CDT		High for illegal opcodes.
CDU	$CD11 \wedge CD10$	N3 ∨ N7 ∨ T3 ∨ T7 ∨ A3 ∨ A7 ∨ L3 ∨ L7 ∨ S3 ∨ S7 ∨ P2 ∨ P4 ∨ D2 ∨ X3 ∨ R1 ∨ R3
CDW	$\overline{CD12} \wedge \overline{CD11}$	N0–1 ∨ T0–1 ∨ A0–1 ∨ L0–1 ∨ S0–1 ∨ M0–1 ∨ X0–1 ∨ B0–7
CDX	$\overline{CD12} \wedge CD11$	N2–3 ∨ T2–3 ∨ A2–3 ∨ L2–3 ∨ S2–3 ∨ R0–3 ∨ D0–2 ∨ P4
CDY	$CD12 \wedge \overline{CD11}$	N4–5 ∨ T4–5 ∨ A4–5 ∨ L4–5 ∨ S4–5 ∨ P1 ∨ P3 ∨ X2 ∨ X5
CDZ	$CD12 \wedge CD11$	N6–7 ∨ T6–7 ∨ A6–7 ∨ L6–7 ∨ S6–7 ∨ P0 ∨ P2 ∨ X3–4

APPENDIX III

CENTRAL PROCESSOR LOGIC SYMBOLS

SYMBOL	NAME	DRAWING LOCATION
A	Accumulator Register	Z, AD, AA, AB, W, N, P, Q
ABR	Addressable Bus Register	ET 3A
AC	Address Counter Register	C, D, E
ACA→G	Computed Signals From AC Register	C, D, E
AD1, 2	Add Delay Flip–flops	AJ
ADD	$\equiv (PS1 \wedge M{-}3) \vee (PS1 \wedge M{-}3)$	AJ4D
AE	Accumulator Exponent Storage Register	K, L
AFC	Finish Cycle, MM–10 Unit A	MJ1F
AM	Address Memory Storage Flip–flops	D, E
AN	Register A Octally Normalized	Q4F
AS2	Accumulator Sign Flip–flop	T3F
AZ	Register A Zero, Computed Signal	AH
AZ1, 2	Register A Zero Flip–flops	AH
B	Memory Buffer Register	HA, B
BA	Buffer Address Register	C, D, E
BB	Bus Busy (to MM–10)	MNIC
BD	Control Panel Bulb Drivers	XC
BF	Bus Free (to MM–10)	MJ5G
BFC	Finish Cycle, MM–10 Unit B	MJ2F
BK	Borrow Kill (Subtractor)	K, L
BLA	Block Line Amplifier	AK1A

SYMBOL	NAME	DRAWING LOCATION
BNA	Bus Not Available (to MM–10)	MN3D
BNF	Bus Not Free (to MM–10)	MJ4G
BP	Borrow Propagate (Subtractor)	K, L
BRA	Branch Flip–flop (Input/Output)	EC1A
BS	Bootstrap Flip–flop	EC3A
BTK	Computed Signal From KC Sequencer	T1A
BU	Bus Register Boosters	LE
CA	Command Address Register	C, D, E
CCF	CC–10 Fault Light	XC3A
CD	Command Decode Register	C, D, E, PD
CDA→Z	CD Register Decoding	BE
CED	Count Exponent Delay	BB4D
CFC	Finish Cycle, MM–10 Unit C	MJ3F
CEN	Count Exponent Register, Negative	BB4A
CEP	Count Exponent Register, Positive	BB4A
CHE	Change Exponent	BB4C
CK1, 2	Clock Frequency Divider (to MM–10)	MK4A
CKA	KA Sequencer Advance, Computed Signal	LJ1B
CKC	"Reset" Signal for Master Control	BX1D
CMP	Complement, Computed Signal	AJ4A
CP	Character Counter (Input/Output)	EC, EB
CPD	Cleanup Phase Done	BA2A
CQ	Character Counter (Input/Output)	EC
CY	Carryout from 41st Stage of Adder	Q3C
CYE	Carryout Enable	BB6B
CZ	Carry in to Zero Stage of Adder	T1D
D	Denominator Register	Z, AD, AA, AB, W, N, P, Q
DA1, 2	Data Available Flip–flops	AC
DAL	Data Available From Line (From MM–10)	MN1D

		DRAWING
SYMBOL	NAME	LOCATION
DAR	Data Available Recognized	AF4D
DAS	Data Available Signal From QM Sequencer	AF5D
DAT	Data Transmit (to MM-10)	MK3A
DAX	Data Available From External Memory	MK2B
DB2	Discarded Bits	T5A
DKD	Done Signal From KD Sequencer	BC6C
DKJ	Done Signal From KJ Sequencer	ET6E
DKW	Done Signal From KW Sequencer	LD4D
DQS	Done Signal From QS Sequencer	S6F
DQW	Done Signal From QW Sequencer	EB4E
DRA	Done Signal From Repeat A and L Opcodes	LJ3C
DRL	Done From Repeat Logic	LJ4B
DRT	Done Signal From Repeat T and S Opcodes	LJ2B
DW	Delay Write Flip-flop	AF5B
EA	Exponent of Accumulator Register	K, L
EAZ	Contents of EA Register = Zero	J2D
EC1, 2	Clock Generation Signals to MM-10	ML
ECF	Enable Command Flags	BT2B
EDF	Enable DATA Flags	BT4E
EE	Exponent Equal, Computed Signal	S3E
EE2	Exponent Equal Flip-flop	T4A
EEA, B	Computed Signals From Exponent Circuit	J
EL1, 2	(EA) $\geq$ (EP) Flip-flops	T
ELF	Enable Logic Flags	BT3E
EP	Exponent of Operand Register	K, L
EPZ	Contents of EP Register = Zero	J1D
ER	Error Signal (Input/Output)	AL4G
ERD	External Memory Ready	MK4B

SYMBOL	NAME	DRAWING LOCATION
ES	Exponent Sum Register	K, L
ESA→Y	Computed Signals From ES Register	J
ESZ	Contents of ES Register = Zero	J2C
EUF	Exponent Underflow	J4C
EXM	External Memory Diagnostic Routine Flip-flop	XC4E
EXN	Exponent Range Normal	JB4
EXT	External Memory	MK5D
EXZ	Exponent Zero	B5F
FC1, 2	Finish Cycle From Memory, Panel F	AC
FC1*, 2*	Finish Cycle From Memory, Panel N	AC
FCA→C	Finish Cycle From MM-10 Units A, B or C	MN
FCX	Finish Cycle From External (MM-10) Memory	MK3D
FM	Finish Memory Operation	MN1D
FS	Forbidden State for Exponent Circuits	J3D
GBA	Access Memory Address in BA Register	AG1A
GCA	Access Memory Address in CA Register	AG4A
GDA→C	Gate Data From MM-10 Units A, B or C	MJ
GKC	Master Control Advance Signal	BD4D
GKL	"GO" KL Sequencer	LF1B
GKM	"GO" KM Sequencer	BY3B
GKP	"GO" KP Sequencer	BY5B
GKT	"GO" KT Sequencer	BZ5F
GKW	"GO" KW Sequencer	LD1G
GKX	"GO" KX Sequencer	BZ4B
GQS	"GO" QS Sequencer	S2D
GQW	"GO" QW Sequencer	EB5A

SYMBOL	NAME	DRAWING LOCATION
GQZ	"GO" QZ Sequencer	KA1A
GRS	"GO" Repeat Sequencer	LD2G
GXC	Start KJ Sequencer	BY4B
GXS	Start KJ Sequencer	BZ5B
H	Line Response Register	LA, LB, LC
HPE	Parity Error (Bit 9 of H Register)	LB5D
HRE	REQ Response to Instruction (Bit 10 of H Register)	LC1D
HTE	Time Error (Bit 11 of H Register)	LC2D
J	Interrupt Request Register	LC
JCA	Command Flag 31	LC5B
JCB	Command Flag 30	LC4B
JDA	Data Flag 31	LC1B
JDB	Data Flag 30	LB5B
JLA	Logic Flag 31	LC3B
JLB	Logic Flag 30	LC2B
JLE	Large Exponent (Operand Too Large)	LB4B
JNC	Non–existent Code	LA1B
JRT	Real Time Reference Interrupt	LB3B
JWA	Input/Output Interrupt Request	LA5B
JWB	Input/Output Interrupt Request	LA4B
JWC	Input/Output Interrupt Request	LA3B
JWD	Input/Output Interrupt Request	LA2B
KA	Add/Subtract Sequencer	LJ
KC	Master Control Sequencer	BD, BE
KD	Divide/Multiply Sequencer	BC
KJ	Jump Sequencer	EJ
KL	Logic Sequencer	LF
KM	Multi–Access Sequencer	LD
KP	Put–away Sequencer	LH
KR1, 2	Done Signals From All Opcodes	BD

		DRAWING
SYMBOL	NAME	LOCATION
KT	Transfer Sequencer	ET
KW	Block Input/Output Opcode Sequencer	EA
KX	Single Character Transmit Sequencer	EC
KZ	States in Master Control Sequencer	BD, BZ
LA	Line Amplifier	AK
LAP	Long Accumulator Put	BC5D
LCE	Leapfrog Carry Extension	W
LD	Line Drivers	AK
LE	Large Exponent	S1A
LES	Recognize Large Exponent	S6A
LFC	Leapfrog Carry	AA, AB, W, N, P
LFK	Leapfrog Kill	AA, AB, W, N, P
LJC	Divide/Multiply Start Signal	BT3H
LKE	Leapfrog Kill Extension	W
LKC	Loop KC (Repeat)	BX1G
LP	Logic Product	AH2E
LPA, C, D	Logic Product to Leapfrog on Panels A, C, and D	Y
LR	Line Receiver Register	AL
LT	Line Transmitter	AK
M	Multiplicand Register	Z, AD, AA, AB, W, N, P, Q
MA	Memory Address Register	C, D, E
MCA	Computed Signals From KC Sequencer	BX, BZ, BY, BS, BT
MF	Memory Finish, Computed Signal	AF5B
MM0, 1	Modulo Three Counter Flip-flops	BB
MNA	Non-existent Memory Location	AF1F
MRA→C	Memory Ready From MM-10 Units A, B, or C	MN
MS	Memory Start	AF5B
MSA→C	Memory Start to MM-10 Units A, B, or C	MH

SYMBOL	NAME	DRAWING LOCATION
MT0, 1	Modulo Three Counter Flip–flops	BB
MWR	Memory Write, External	MJ
MY, Z	Memory Timer Flip–flops	AC
N	Numerator Register	Z,AD,AA,AB,W, N,P,Q
NN	N Register Octally Normalized, Computed Signal	Q4B
NN2	N Register Octally Normalized Flip–flop	T4F
NRC	Illegal Opcode Signal	ET4E
PA	Memory Parity Checking Circuits	HB
PAA→D	Input/Output Parity Check Circuits	AL
PBL	Product Bits Lost	BC4F
PC	Propagate Carry (Adder)	Z,AD,AA,AB,W, N,P,Q
PCP	Product Cleanup Phase	AJ3A
PE	Pickapoint Exponent Register	LA,LB
PED	Input/Output Parity Error Detected	AL5F
PGA→D	Input/Output Parity Generate Circuits	AK
PK	Propagate Kill (Adder)	Z,AD,AA,AB,W, N,P,Q
PMF	Memory Parity Error (To Indicator Lights)	PF1C
PS1, 2	Product Sign Flip–flops	AJ
PSC	Prohibit Memory Start	XC3C
PW	Parity Wrong Memory	HB3E
QA	Accumulator Put Sequencer	BF
QB	External Memory Control Sequencer	MK
QC	Command Access Sequencer	LD
QM	Memory Control Sequencer (Internal)	AF
QP	Product Sequencer	AJ
QQ	Quotient Sequencer	BA
QS	Sum/Difference Sequencer	S

SYMBOL	NAME	DRAWING LOCATION
QW	Input/Output Sequencer	EB
QZ	Zero Exponent Sequencer	KA
R	Adder	Z, AD, AA, AB, W N, P, Q
RAK	Start QA Sequencer	BT4H
RD2	External Memory Ready	MK3C
RDX	Read Driver, X Lines	FA1B
RDY	Read Driver, Y Lines	FA1F
RE	KC Sequencer Advance, Computed Signals	BX
REM	Read External Memory	MN4C
REQ	Request for Character (Received)	AL3E
RET	Return Flip–flop (Input/Output)	EC4A
RKM	Ready From KM Sequencer	LD4E
RL1, 2	Reset LR	AL
RLR	Reset LR Register	AL1D
RM	Read Memory Signal	AC1B
RML	Start KC Sequencer, Mode 0 and 1	BT3H
RQM	Ready QM Sequencer	AF6D
RQS	Ready QS Sequencer	S5F
RQW	Ready QW Sequencer	EB5A
RS1, 2	Remainder Sign Flip–flops	BA
RTA→C	Real Time Signals	AK
RTM	Read Transmitting Memory	MN4A
RTF	$\overline{\text{RTJ}}$	LC1E
RTJ	Terminate Repeat Operation	LF5A
RTR	Real Time Signals Flip–flop	AK5A
RTS	Real Time Signal	AK5A
RU	Round Up (Sum/Difference)	T1A
RWA, B	Echo Flip–flops for UWA, B	LA
S	Sum Register	Z, AD, AA, AB, W, N, P, Q

SYMBOL	NAME	DRAWING LOCATION
SA	Sense Amplifier	FB
SC	Subtractor–Comparator	K, L
SCA	Send CA Register to Bus Register	LE1A
SE	Master Control Advance Signal	BX
SE1→3	Start External Memory	MH
SHR	Send H Register to Bus Register	LE4A
SHS	Shift–Send Signal (Input/Output)	EC2A
SHT	Shift Right Three	AJ3D
SJR	Send J Register to Bus Register	LE3A
SKC	Advance Start to KC sequencer	BS
SKJ	Start KJ Sequencer	BZ5F
SKM	Start KM Sequencer	BY4B
SKP	Start KP Sequencer	BZ4F
SM1, 2	Do a Sum Operation Flip–flop	S
SMA→H	Select Memory A Through H, Computed Signals	AF
SN	S Register Normalized	Q5D
SPE	Select Register PE	ET5A
SRH	Select Register H	ET4B
SRJ	Select Register J	ET3B
SRU	Select Register U	ET4B
ST	Strobe Read Inverters	AC6A
SUR	Send U Register to Bus Register	LE2A
SW	$\Rightarrow S42 \vee S43 \vee S44$	Q6D
SWA→D	Interrupt Synchronizer Flip–flops	LA
SY1, 2	Input/Output Synchronization Flip–flops	LA
SYA→D	States of Input/Output Synchronization	AL
SZ	S Register Zero, Computed Signal	AH3B
SZ1, 2	S Register Zero Flip–flops	AH
SZA→G	S Register Zero, Computed Signals	AH

SYMBOL	NAME	DRAWING LOCATION
TCA	Start Time Out (1 Second)	AK5D
TCW	Transmit Complete Instruction	AF3A
TDA	Transmit Data	EB5E
TE1, 2	Time of Execute (CC–10) Flip–flops	AK5E
TED	Time Error Signal	AK5E
TKC	"Tilt" KC Sequencer	BX4D
TL1, 2	Mantissa Too Large Flip–flops	LH
TLA	Mantissa Too Large	AH3A
TR	Time of Read	AC1D
TRF	Transmit Request For Character	AK2G
TRX, Y	Time of Read, X–Y Lines	FA
TSB	Transmit 6 Bits	AK2G
TW	Time of Write	AC
TWA→D	Input/Output Interrupt Buffer	LA6A
TWX, Y	Time of Write, X–Y Lines	FA
UCA	Command Flag 31	LC5C
UCB	Command Flag 30	LC4C
UDA	Data Flag 31	LC1C
UDB	Data Flag 30	LB5C
UJE	Jump Enable	LA1C
ULA	Logic Flag 31	LC3C
ULB	Logic Flag 30	LC2C
UPE	Pickapoint Mode	LB4C
URB	Ring Bell	LB1C
URT	Real Time Reference Enable	LB3C
UWA	Transmit Interrupt Request	LA5C
UWB	Transmit Interrupt Request	LA4C
UWC, D	Reserved Bits in U Register	LA3C
WCA	KW Advance Signal	EA3F
WDX, Y	Write Drivers, X–Y Lines	FA1C
WG1, 2	Waiting to Use QS Sequencer	S

		DRAWING
SYMBOL	NAME	LOCATION
WM	Write Memory Signal	AC2B
WR1, 2	Write Flip–flops (Memory Control)	AF
WS1, 2	Working Sign Flip–flops	T
XCM	Diagnostic Switch on Control Panel	PR3C
XJJ	⇒ Interrupt	LC5E
XL	X Decoding, Left Hand Bits	AC
XPC	External Parity Control	PR3C
XR	X Decoding, Right Hand Bits	AC
XRC	Diagnostic Switch on Control Panel	PR3C
XZ	Index Equals Zero	ET2A
XSD	Memory Block Out Switch on Control Panel	PR3C
YL	Y Decoding, Left Hand Bits	AC
YR	Y Decoding, Right Hand Bits	AC
ZB	Zero Order Borrow	J3F
ZBA	AC Register Equals Zero	E4D
ZE2	Zero Exponent, From QS Sequencer	S4F
ZM	Zero Machine	PH5B
ZTM	Zeros To Memory	AJ5C

APPENDIX IV

INDICATOR LIGHT CONNECTIONS FOR CENTRAL PROCESSOR LOGIC SYMBOLS

	Signal				Signal		
Signal	Location	Pin	Print	Signal	Location	Pin	Print
0(0)CQ	LA51	F	EC1B	A11	AA49	T	AA3F
1(0)CQ	LA51	H	EC2B	A12	CA49	A	AA4F
2(0)CQ	LA51	J	EC1B	A13	CA49	B	AA5F
7EA0(L21)28B21	HA51	R	F4D	A14	CA49	C	AA6F
14M0(0)14B0	HA51	K	F1B	A15	CA49	D	AB1F
20M15(0)20B15	HA51	L	F2B	A16	CA49	E	AB2F
26M21(0)26B21	HA51	M	F3B	A17	CA49	F	AB3F
41AZ0	BA51	P	AH1D	A18	CA49	H	AB4F
64(0)CA	JA56	C	AG5D	A19	CA49	J	AB5F
A–1	AA49	D	Z3F	A20	CA49	K	AB6F
A0	AA49	E	Z4F	A21	CA49	L	W1F
A1	AA49	F	Z5F	A22	CA49	M	W2F
A2	AA49	H	Z6F	A23	CA49	N	W3F
A3	AA49	J	AD1F	A24	CA49	P	W4F
A4	AA49	K	AD2F	A25	CA49	R	W5F
A5	AA49	L	AD3F	A26	CA49	S	W6F
A6	AA49	M	AD4F	A27	DA49	A	N1F
A7	AA49	N	AD5F	A28	DA49	B	N2F
A8	AA49	P	AD6F	A29	DA49	C	N3F
A9	AA49	R	AA1F	A30	DA49	D	N4F
A10	AA49	S	AA2F	A31	DA49	E	N5F

	Signal				Signal		
Signal	Location	Pin	Print	Signal	Location	Pin	Print
A32	DA49	F	N6F	AD1	BA51	F	E5D
A33	DA49	H	P1F	AD2	BA51	H	AJ5F
A34	DA49	J	P2F	AE0	KA50	A	K1A
A35	DA49	K	P3F	AE1	KA50	B	K3A
A36	DA49	L	P4F	AE2	KA50	C	K4A
A37	DA49	M	P5F	AE3	KA50	D	K5A
A38	DA49	N	P6F	AE4	KA50	E	L1A
A39	DA49	P	Q1F	AE5	KA50	F	L3A
A40	DA49	R	Q2F	AE6	KA50	H	L4A
A41	DA49	S	Q3F	AE(0)EA	KA53	C	M3F
A(0)N	CA54	N	Y1A	AM8	JA56	K	D4F
A(L1)M	DA54	M	X3D	AM9	JA56	L	D5F
A(R3)M	DA54	N	X5D	AM10	JA56	M	E1F
AC0	JA51	A	C1E	AM11	JA56	N	E2F
AC1	JA51	B	C2E	AM12	JA56	P	E3F
AC2	JA51	C	C3E	AM(0)CA	JA56	B	AG3D
AC3	JA51	D	C4E	AS2	KA51	F	T3F
AC4	JA51	E	C5E	AS2(0)SM1	KA54	B	T4B
AC5	JA51	F	D1E	AS2(C)SM1	KA54	C	T5B
AC6	JA51	H	D2E	AS2(0)WS1	KA53	R	T1B
AC7	JA51	J	D3E	AZ1	BA54	A	AH1E
AC8	JA51	K	D4E	AZ2	BA54	B	AH1F
AC9	JA51	L	D5E	B0	HA49	E	HA1A
AC10	JA51	M	E1E	B1	HA49	F	HA2A
AC11	JA51	N	E2E	B2	HA49	H	HA3A
AC12	JA51	P	E3E	B3	HA49	J	HA4A
AC13	JA51	R	E4E	B4	HA49	K	HA1B
AC14	JA51	S	E5E	B5	HA49	L	HA2B
AC(+1)BA	JA55	F	AG1C	B6	HA49	M	HA3B
AC(+1)CA	JA54	F	AG1D	B7	HA49	N	HA4B

Signal	Signal Location	Pin	Print	Signal	Signal Location	Pin	Print
B8	HA49	P	HA1D	B(L21)D	CA54	D	U5D
B9	HA49	R	HA2D	B(R15)CD	JA56	F	AG2F
B10	HA49	S	HA3D	B(R21)EP	KA54	L	M2C
B11	HA49	T	HA4D	B(R21)EP	KA54	M	M3C
B12	HA50	A	HA1E	BA0	JA50	A	C1C
B13	HA50	B	HA2E	BA1	JA50	B	C2 C
B14	HA50	C	HA3E	BA2	JA50	C	C3C
B15	HA50	D	HA1F	BA3	JA50	D	C4C
B16	HA50	E	HA2F	BA4	JA50	E	C5C
B17	HA50	F	HA3F	BA5	JA50	F	D1C
B18	HA50	H	B1B	BA6	JA50	H	D2C
B19	HA50	J	B3B	BA7	JA50	J	D3C
B20	HA50	K	B4B	BA8	JA50	K	D4C
B21	HA50	L	B5B	BA9	JA50	L	D5C
B22	HA50	M	B1C	BA10	JA50	M	E1C
B23	HA50	N	B3C	BA11	JA50	N	E2C
B24	HA50	P	B4C	BA12	JA50	P	E3C
B25	HA50	R	B5C	BA13	JA50	R	E4C
B26	HA50	S	B1E	BA14	JA50	S	E5C
B27	HA51	A	B3E	BA15	JA55	N	E5C
B28	HA51	B	B4E	BA(0)AC	JA55	A	AG2B
B29	HA51	C	B5E	BA(0)MA	JA54	A	AG1B
B30	HA51	D	B1F	BLA	MA49	J	AK1A
B31	HA51	E	B3G	$\overline{\text{BLA}}$	MA49	K	AK1A
B32	HA51	F	B4F	$\overline{\text{BLA}}$ *1	MA51	A	AK1A
B(0)BA	JA56	H	AG3C	$\overline{\text{BLA}}$ *2	MA51	B	AK1A
B(0)D 14/0	CA54	E	U1B	$\overline{\text{BLA}}$ *3	MA51	C	AK1A
B(0)D 20/15	CA54	F	U2B	$\overline{\text{BLA}}$ *4	MA51	D	AK1A
B(0)D 26/21	CA54	H	U2B	BRA	LA51	P	EC1A
B(0)D 31/27	CA54	J	U3B	BS	LA51	A	EC3A

Signal	Signal Location	Pin	Print	Signal	Signal Location	Pin	Print
BU0	EA51	A	LE1C	CA7	JA52	J	D3D
BU1	EA51	B	LE2C	CA8	JA52	K	D4D
BU2	EA51	C	LE3C	CA9	JA52	L	D5D
BU3	EA51	D	LE4C	CA10	JA52	M	E1D
BU4	EA51	E	LE6C	CA11	JA52	N	E2D
BU5	EA51	F	LE2D	CA12	JA52	P	E3D
BU6	EA51	H	LE3D	CA13	JA52	R	E4D
BU7	EA51	J	LE4D	CA14	JA52	S	E5D
BU8	EA51	K	LE5D	CA(0)AC	JA55	B	AG5B
BU9	EA51	L	LE6D	CA(0)AM	JA56	A	AG1F
BU10	EA51	M	LE1F	CA(0)MA	JA54	B	AG4B
BU11	EA51	N	LE3F	CD0	JA53	A	C1F
BU12	EA51	P	LE4F	CD1	JA53	B	C2F
BU13	EA51	R	LE5F	CD2	JA53	C	C3F
BU14	EA51	S	LE6F	CD3	JA53	D	C4F
BU(0)D	CA54	A	U6F	CD4	JA53	E	C5F
C1	BA50	P	AH6C	CD5	JA53	F	D1F
C1–	LA52	P		CD6	JA53	H	D2F
C2	BA50	R	AH6C	CD7	JA53	J	D3F
C2–	LA53	P		CD8	JA53	K	D4F
C2∧Clear LR	MA52	T	AL4C	CD9	JA53	L	D5F
C2∧$\overline{\text{CKC}}$	BA52	E	BX1E	CD10	JA53	M	E1F
C2∧GKL	LA54	E	LF1C	CD11	JA53	N	E2F
CA0	JA52	A	C1D	CD12	JA53	P	E3F
CA1	JA52	B	C2D	CD13	JA53	R	E4F
CA2	JA52	C	C3D	CD14	JA53	S	E5F
CA3	JA52	D	C4D	CD(0)BA	JA56	J	AG5C
CA4	JA52	E	C5D	CDA	HA52	A	BE1D
CA5	JA52	F	D1D	CDC	HA52	C	BE2D
CA6	JA52	H	D2D	CDD	HA52	D	BE3D

Signal	Signal Location	Pin	Print	Signal	Signal Location	Pin	Print
CDE	HA52	E	BE4D	CYE	BA54	F	BB6B
CDG	HA52	F	BE5E	CZ1	KA51	A	T1D
CDH	HA52	H	BE5D	CZ2	KA51	B	T1F
CDK	HA52	K	BE3G	D–1	AA50	D	Z3A
CDL	HA52	L	BE4E	D–2	AA50	C	Z2A
CDN	HA52	N	BE6D	D–3	AA50	B	Z1A
CDR	HA52	R	BE2G	D–4	AA50	A	Z1B
CDT	HA52	T	BE3H	D0	AA50	E	Z4A
CED	BA51	E	BB4D	D1	AA50	F	Z5A
$\overline{\text{CKA}}$	LA55	P	LJ1B	D2	AA50	H	Z6A
CKC *	BA52	H	BX1D	D3	AA50	J	AD1A
CKC *	BA52	J	BX2D	D4	AA50	K	AD2A
Clear 5D0	JA56	R	U5B	D5	AA50	L	AD3A
Clear 14D0	AA54	B	U5B	D6	AA50	M	AD4A
Clear 41D15	AA54	C	U3B	D7	AA50	N	AD5A
Clear 41N32	AA54	D	Y5A	D8	AA50	P	AD6A
Clear AE	KA53	B	M4A	D9	AA50	R	AA1A
Clear B	HA51	S	AH6E	D10	AA50	S	AA2A
Clear EA	KA53	J	M2A	D11	AA50	T	AA3A
Clear EP	KA54	H	M1A	D12	CA50	A	AA4A
Clear ES	KA53	N	M3A	D13	CA50	B	AA5A
Clear N	CA54	P	Y3A	D14	CA50	C	AA6A
Clear S	CA54	R	R4B	D15	CA50	D	AB1A
CP1	LA51	L	EC1E	D16	CA50	E	AB2A
CP2	LA51	M	EC2E	D17	CA50	F	AB3A
CP3	LA51	N	EC2E	D18	CA50	H	AB4A
CP(+1)CQ	LA51	E	EC1B	D19	CA50	J	AB5A
CQ1	LA51	B	EC1C	D20	CA50	K	AB6A
CQ2	LA51	C	EC1C	D21	CA50	L	W1A
CQ3	LA51	D	EC2C	D22	CA50	M	W2A

Signal	Signal Location	Pin	Print	Signal	Signal Location	Pin	Print
D23	CA50	N	W3A	DKD	BA50	N	BC6D
D24	CA50	P	W4A	DQS	KA56	B	S6F
D25	CA50	R	W5A	DQW	LA50	A	EB4E
D26	CA50	T	W6A	DRA	LA54	P	LJ3C
D27	DA50	A	N1A	DRL	LA54	K	LJ4B
D28	DA50	B	N2A	DW	JA55	L	AF5B
D29	DA50	C	N3A	EA0	KA49	A	K1B
D30	DA50	D	N4A	EA1	KA49	B	K3B
D31	DA50	E	N5A	EA2	KA49	C	K4B
D32	DA50	F	N6A	EA3	KA49	D	K5B
D33	DA50	H	P1A	EA4	KA49	E	L1B
D34	DA50	J	P2A	EA5	KA49	F	L3B
D35	DA50	K	P3A	EA6	KA49	H	L4B
D36	DA50	L	P4A	EA7	KA49	J	L5B
D37	DA50	M	P5A	EA(0)AE	KA53	A	M4F
D38	DA50	N	P6A	EA(0)ES	KA53	K	M2D
D39	DA50	P	Q1A	EAZ	KA52	A	J2D
D40	DA50	R	Q2A	EE2	KA51	J	T4A
D41	DA50	S	Q3A	EL1	KA52	K	T5C
D(0)N	CA54	K	Y4C	EL2	KA52	L	T5F
D(0)S	DA54	H	R3F	$\overline{\text{END(0)LD}}$	MA54	K	AK1G
D(L2)S	DA54	F	R5E	EP0	KA49	K	K1D
DA1	FA50	N	AC3F	EP1	KA49	L	K3D
DA2	FA50	H	AC3F	EP2	KA49	M	K4D
DAR	JA55	J	AF4D	EP3	KA49	N	K5D
DAS	JA55	S	AF5D	EP4	KA49	P	L1D
DATA$\wedge\overline{\text{PED}}$	MA50	D	AL4G	EP5	KA49	R	L3D
DAX	MA50	L	MM1E	EP6	KA49	S	L4D
DAY	MA50	M	MM1F	EP7	KA49	T	L5D
DB2	KA51	E	T6A	EP(0)ES	KA53	L	M1A

Signal	Signal Location	Pin	Print	Signal	Signal Location	Pin	Print
EPZ	KA52	C	J1D	GKT	BA53	J	BZ5F
ER	MA50	E	AL4G	GKW	LA49	K	LD4G
ER *	LA51	T	EB6B	GKX	BA53	E	BZ4B
ES0	KA50	K	K1F	$\overline{\text{GQW}}$	LA50	B	EB5A
ES1	KA50	L	K3F	GRN∧$\overline{\text{PED}}$	MA50	P	AL2G
ES2	KA50	M	K4F	GRS	LA52	B	LD1G
ES3	KA50	N	K5F	GXC	BA53	C	BY4B
ES4	KA50	P	L1F	GXS	BA53	F	BZ5B
ES5	KA50	R	L3F	H0	EA52	A	LA1D
ES6	KA50	S	L4F	H1	EA52	B	LA2D
ES7	KA50	T	L5F	H2	EA52	C	LA3D
ES(0)EA	KA53	D	M4D	H3	EA52	D	LA4D
ES(+1)EA	KA53	F	M5D	H4	EA52	E	LA5D
ES(−1)EA	KA53	H	M1D	H5	EA52	F	LB1D
ES(+1)EP	KA54	J	M5C	H6	EA52	H	LB2D
ES(−1)EP	KA54	K	M4C	H7	EA52	J	LB3D
ES(C)EA	KA53	E	M3D	H8	EA52	K	LB4D
ESZ	KA52	B	J2C	H9	EA52	L	LB5D
EXZ	HA49	D	B5F	H12	EA52	P	LC3D
FC1	FA50	J	AC4D	H13	EA52	R	LC4D
FC2	FA50	P	AC4F	H14	EA52	S	LC5D
FCX	MA50	N	MM2E	HPE	EA52	M	LC1D
FCY	MA50	P	MM2F	HTE	EA52	N	LC2D
FS	KA52	E	J3D	J5	EA50	F	LB1B
GBA	JA55	R	AG2A	J6	EA50	H	LB2D
GCA	JA55	P	AG4A	JCA	EA50	R	LC5B
GKC	BA52	C	BD4E	JCB	EA50	S	LC4B
$\overline{\text{GKC}}$	LA54	B	BD4D	JDA	EA50	M	LC1B
GKM	BA53	A	BY3B	JDB	EA50	L	LB5D
GKP	BA53	D	BY5B	JLA	EA50	P	LC3B

Signal	Signal Location	Pin	Print	Signal	Signal Location	Pin	Print
JLB	EA50	N	LC2B	KDH^DQQ	BA49	P	BA2B
JLE	EA50	K	LB4D	KJ1	EA54	C	EJ1C
JNC	EA50	A	LA1B	KJ2	EA53	C	EJ1F
JRT	EA50	J	LB3B	KJ3	EA54	D	EJ3C
JWA	EA50	E	LA5B	KJ4	EA53	D	EJ2F
JWB	EA50	D	LA4B	KJ5	EA54	E	EJ5C
JWC	EA50	C	LA3B	KJ6	EA53	E	EJ5F
JWD	EA50	B	LA2B	KL1	LA54	F	LF1C
KA1	LA54	L	LJ2D	KL2	LA55	F	LF1F
KA2	LA55	L	LJ2F	KL3	LA54	H	LF2C
KA3	LA54	M	LJ3D	KL4	LA55	H	LF2F
KA4	LA55	M	LJ3F	KM1	LA52	F	LD1C
KA5	LA54	N	LJ4D	KM2	LA53	F	LD1E
KA6	LA55	N	LJ4F	KM3	LA52	H	LD2C
KC1	BA52	L	BD1B	KM4	LA53	H	LD2E
KC2	BA53	L	BE1B	KM5	LA52	J	LD3C
KC3	BA52	M	BD3B	KM6	LA53	J	LD3E
KC4	BA53	M	BE3B	KP1	LA52	L	LH1C
KC5	BA52	N	BD4B	KP2	LA53	L	LH1E
KC6	BA53	N	BE4B	KP3	LA52	M	LH2C
KC7	BA52	P	BD6B	KP4	LA53	M	LH2E
KC8	BA53	P	BE5B	KP5	LA52	N	LH4C
KD1	BA49	F	BC1D	KP6	LA53	N	LH3E
KD2	BA50	F	BC1F	KR1	BA52	A	BD2K
KD3	BA49	H	BC2D	KR2	BA52	B	BD3J
KD4	BA50	H	BC2F	KT1	EA54	A	ET1D
KD5	BA49	J	BC3D	KT2	EA53	A	ET1F
KD6	BA50	J	BC3F	KT3	EA54	B	ET3D
KD7	BA49	K	BC3D	KT4	EA53	B	ET3F
KD8	BA50	K	BC4F	KW1	LA49	L	EA1B

	Signal				Signal		
Signal	Location	Pin	Print	Signal	Location	Pin	Print
KW2	LA50	L	EA1D	LP	AA54	A	AH2E
KW3	LA49	M	EA3B	LP	BA54	E	AH2E
KW4	LA50	M	EA3D	LR0	MA52	E	AL1A
KW5	LA49	N	EA5B	$\overline{LR0}$	MA53	E	AL1A
KW6	LA50	N	EA5D	LR1	MA52	F	AL2A
KW7	LA49	P	EA6B	$\overline{LR1}$	MA53	F	AL2A
KW8	LA50	P	EA6D	LR2	MA52	H	AL3A
KX1	LA49	F	EC3C	$\overline{LR2}$	MA53	H	AL3A
KX2	LA50	F	EC3E	LR3	MA52	J	AL4A
KX3	LA49	H	EC4C	$\overline{LR3}$	MA53	J	AL4A
KX4	LA50	H	EC4E	LR4	MA52	K	AL5A
KX5	LA49	J	EC5C	$\overline{LR4}$	MA53	K	AL5A
KX6	LA50	J	EC5E	LR5	MA52	L	AL1B
LAP	BA51	R	BC5D	$\overline{LR5}$	MA53	L	AL1B
LD0	MA51	E	AK2A	LR6	MA52	M	AL2B
LD1	MA51	F	AK3A	$\overline{LR6}$	MA53	M	AL2B
LD2	MA51	H	AK4A	LR7	MA52	N	AL3B
LD3	MA51	J	AK5A	$\overline{LR7}$	MA53	N	AL3B
LD4	MA51	K	AK1B	LR8	MA52	P	AL4B
LD5	MA51	L	AK2B	$\overline{LR8}$	MA53	P	AL4B
LD6	MA51	M	AK3B	LR9	MA52	R	AL5B
LD7	MA51	N	AK4B	$\overline{LR9}$	MA53	R	AL5B
LD8	MA51	P	AK5B	$\overline{LR10}$	MA53	S	
LD9	MA51	R	AK1D	LR(0)S	AA53	A	R5B
LD10	MA51	S	AK2D	LR(0)S	AA53	B	R5B
LD11	MA51	T	AK3D	M–1	AA51	B	Z1E
LE	KA52	H	S1A	M–2	AA51	C	Z2E
LES	KA52	F	S6A	M–3	AA51	D	Z3E
LKC	BA52	F	BX1G	M0	AA51	E	Z4E
$\overline{LKC}$	LA54	C	BX1G	M1	AA51	F	Z5E

Signal	Signal Location	Pin	Print	Signal	Signal Location	Pin	Print
M2	AA51	H	Z6E	M32	DA51	F	N6E
M3	AA51	J	AD1E	M33	DA51	H	P1E
M4	AA51	K	AD2E	M34	DA51	J	P2E
M5	AA51	L	AD3E	M35	DA51	K	P3E
M6	AA51	M	AD4E	M36	DA51	L	P4E
M7	AA51	N	AD5E	M37	DA51	M	P5E
M8	AA51	P	AD6E	M38	DA51	N	P6E
M9	AA51	R	AA1E	M39	DA51	P	Q1E
M10	AA51	S	AA2E	M40	DA51	R	Q2E
M11	AA51	T	AA3E	M41	DA51	S	Q3E
M12	CA51	A	AA4E	M(0)31B27	HA51	N	F3B
M13	CA51	B	AA5E	M(0)A	DA54	R	X5B
M14	CA51	C	AA6E	M(L2)A	DA54	P	X3F
M15	CA51	D	AB1E	MA0	JA49	A	C1B
M16	CA51	E	AB2E	MA1	JA49	B	C2B
M17	CA51	F	AB3E	MA2	JA49	C	C3B
M18	CA51	H	AB4E	MA3	JA49	D	C4B
M19	CA51	J	AB5E	MA4	JA49	E	C5B
M20	CA51	K	AB6E	MA5	JA49	F	D1B
M21	CA51	L	W1E	MA6	JA49	H	D2B
M22	CA51	M	W2E	MA7	JA49	J	D3B
M23	CA51	N	W3E	MA8	JA49	K	D4B
M24	CA51	P	W4E	MA9	JA49	L	D5B
M25	CA51	R	W5E	MA10	JA49	M	E1B
M26	CA51	S	W6E	MA11	JA49	N	E2B
M27	DA51	A	N1E	MA12	JA49	P	E3B
M28	DA51	B	N2E	MA13	JA49	R	E4B
M29	DA51	C	N3E	MA14	JA49	S	E5B
M30	DA51	D	N4E	MF1	MA50	S	MM4F
M31	DA51	E	N5E	MF^C2*1	JA54	J	AF5B

	Signal				Signal		
Signal	Location	Pin	Print	Signal	Location	Pin	Print
MM0	BA51	C	BB1D	N21	CA52	L	W1B
MM1	BA51	D	BB2D	N22	CA52	M	W2B
MNA *1	JA54	H	AF1F	N23	CA52	N	W3B
MS1	MA50	R	MM3F	N24	CA52	P	W4B
MS⌒C2	JA54	K	AF5B	N25	CA52	R	W5B
MT0	BA51	A	BB1B	N26	CA52	S	W6B
MT1	BA51	B	BB2B	N27	DA52	A	N1B
MY	FA50	L	AC3B	N28	DA52	B	N2B
MZ	FA50	M	AC4B	N29	DA52	C	N3B
N0	AA52	E	Z4B	N30	DA52	D	N4B
N1	AA52	F	Z5B	N31	DA52	E	N5B
N2	AA52	H	Z6B	N32	DA52	F	N6B
N3	AA52	J	AD1B	N33	DA52	H	P1B
N4	AA52	K	AD2B	N34	DA52	J	P2B
N5	AA52	L	AD3B	N35	DA52	K	P3B
N6	AA52	M	AD4B	N36	DA52	L	P4B
N7	AA52	N	AD5B	N37	DA52	M	P5B
N8	AA52	P	AD6B	N38	DA52	N	P6B
N9	AA52	R	AA1B	N39	DA52	P	Q1B
N10	AA52	S	AA2B	N40	DA52	R	Q2B
N11	AA52	T	AA3B	N41	DA52	S	Q3B
N12	CA52	A	AA4B	N(0)BA	JA56	D	AG2C
N13	CA52	B	AA5B	N(0)CA	JA56	E	AG3D
N14	CA52	C	AA6B	N(0)PE	EA54	J	LG5A
N15	CA52	D	AB1B	N(F)H	EA54	P	LG2A
N16	CA52	E	AB2B	N(F)J	EA54	M	LG3A
N17	CA52	F	AB3B	N(F)U	EA54	K	LG4A
N18	CA52	H	AB4B	N(L1)S	DA54	E	R2B
N19	CA52	J	AB5B	N(L3)S	DA54	D	R5F
N20	CA52	K	AB6B	N(T)H	EA54	R	LG1A

	Signal				Signal		
Signal	Location	Pin	Print	Signal	Location	Pin	Print
N(T)J	EA54	N	LG2A	QA4	BA50	M	BF2E
N(T)U	EA54	L	LG4A	QAE∧$\overline{\text{KDC}}$	BA49	R	BF2D
PA16	HA51	H	HB1E	QC1	LA52	C	LD4C
PAA	MA54	J	AL1G	QC2	LA53	C	LD4E
PAB	MA54	H	AL2G	QC3	LA52	D	LD5C
PAC	MA54	F	AL3G	QC4	LA53	D	LD5E
PAD	MA54	E	AL4G	QC5	LA52	E	LD6C
PBL	BA51	P	BC4F	QC6	LA53	E	LD6E
PE0	EA55	A	LA1E	QM1	JA54	C	AF1B
PE1	EA55	B	LA2E	QM2	JA55	C	AF1D
PE2	EA55	C	LA3E	QM3	JA54	D	AF2B
PE3	EA55	D	LA4E	QM4	JA55	D	AF2D
PE4	EA55	E	LA5E	QM5	JA54	E	AF3B
PE5	EA55	F	LB1E	QM6	JA55	E	AF3D
PE6	EA55	H	LB2E	QP1	BA49	C	AJ1C
PE(0)EP	KA54	N	M5A	QP2	BA50	C	AJ1F
PE(C)EP	KA54	P	M1C	QP3	BA49	D	AJ2C
PED	MA50	F	AL5F	QP4	BA50	D	AJ2F
$\overline{\text{PED}}$	MA50	H	AL5G	QP5	BA49	E	AJ3C
PGA	MA54	D	AK4F	QP6	BA50	E	AJ3F
PGB	MA54	B	AK3F	QQ1	BA49	A	BA1C
PGC	MA54	C	AK1F	QQ2	BA50	A	BA1E
PGD	MA54	A	AK5F	QQ3	BA49	B	BA2C
PS1	BA51	K	AJ4C	QQ4	BA50	B	BA2E
PS2	BA51	L	AJ4F	QQZ	KA56	K	
PSC	HA53	A	XC2B	QS1	KA55	C	S1C
$\overline{\text{PW}}$	HA51	J	HB3E	QS2	KA56	C	S1F
QA1	BA49	L	BF1C	QS3	KA55	D	S2C
QA2	BA50	L	BF1E	QS4	KA56	D	S2F
QA3	BA49	M	BF2C	QS5	KA55	E	S3G

Signal	Signal Location	Pin	Print	Signal	Signal Location	Pin	Print
QS6	KA56	E	S3F	RKM	LA52	K	LD4E
QS8	KA56	F	S1E	RL1	MA50	J	AL4C
QW1	LA49	C	EB1B	RL2	MA50	K	AL4E
QW2	LA50	C	EB1D	RLR	MA49	R	AL4D
QW3	LA49	D	EB2C	RLR *	MA49	S	AL4D
QW4	LA50	D	EB2D	RM	FA50	C	AC1B
QW5	LA49	E	EB4C	RQM	JA55	M	AF6D
QW6	LA50	E	EB4D	RQS	KA56	A	S5F
QZ1	KA55	L	KA1C	$\overline{RQW}$	LA50	K	EB5A
QZ2	KA56	L	KA1F	RS1	BA51	M	BA3C
QZ3	KA55	M	KA3C	RS2	BA51	N	BA3E
QZ4	KA56	M	KA2F	RTA	MA49	A	AK5A
QZ5	KA55	N	KA4C	$\overline{RTA}$	MA49	B	AK5A
QZ6	KA56	N	KA4F	RTB	MA49	C	AK5B
R(0)S	DA54	K	R2D	$\overline{RTB}$	MA49	D	AK5B
R(L1)S	DA54	J	R2C	RTC	MA49	E	AK5C
RE1	BA55	A	BX6D	$\overline{RTC}$	MA49	F	AK5C
RE3	BA55	C	BX2G	RTR	MA49	H	AK5A
RE4	BA55	D	BX2G	RU	KA51	P	T1A
RE5	BA55	E	BX3G	RU(0)CZ1	KA51	L	T4C
RE7	BA55	F	BX4G	RWA	EA53	R	LA5C
RE8	BA55	H	BX5G	RWB	EA53	P	LA4C
RE10	BA55	J	BX5G	S—1	AA53	D	Z3D
Ready$\wedge\overline{PED}$	MA50	C		S0	AA53	E	Z4D
REQ	MA50	A	AL3E	S1	AA53	F	Z5D
REQ(0)LD	MA54	M	AK2G	S2	AA53	H	Z6D
Reset QS	KA56	H	T6C	S3	AA53	J	AD1D
Reset SM1	KA54	F	T4C	S4	AA53	K	AD2D
Reset WS1	KA53	S	T2B	S5	AA53	L	AD3D
RET	LA51	S	EC4A	S6	AA53	M	AD4D

	Signal				Signal		
Signal	Location	Pin	Print	Signal	Location	Pin	Print
S7	AA53	N	AD5D	S37	DA53	M	P5D
S8	AA53	P	AD6D	S38	DA53	N	P6D
S9	AA53	R	AA1D	S39	DA53	P	Q1D
S10	AA53	S	AA2D	S40	DA53	R	Q2D
S11	AA53	T	AA3D	S41	DA53	S	Q3D
S12	CA53	A	AA4D	S42	DA54	A	Q4D
S13	CA53	B	AA5D	S43	DA54	B	Q5D
S14	CA53	C	AA6D	S44	DA54	C	Q6D
S15	CA53	D	AB1D	S(0)N	CA54	L	Y2E
S16	CA53	E	AB2D	S(C)N	CA54	M	Y2C
S17	CA53	F	AB3D	S(R3)D	CA54	B	U2D
S18	CA53	H	AB4D	S(R3C)D	CA54	C	U2F
S19	CA53	J	AB5D	S(R3)M	DA54	L	X2B
S20	CA53	K	AB6D	SC(0)ES	KA53	M	M2F
S21	CA53	L	W1D	SCA	EA55	K	LE1A
S22	CA53	M	W2D	SE1	BA55	B	BX2G
S23	CA53	N	W3D	Set SM1	KA54	E	T3C
S24	CA53	P	W4D	SHR	EA55	N	LE4A
S25	CA53	R	W5D	SHS	LA51	R	EC2A
S26	CA53	S	W6D	SHW	BA51	J	AJ6F
S27	DA53	A	N1E	SJR	EA55	M	LE3A
S28	DA53	B	N2E	$\overline{\text{SKC}}$	HA49	A	BS3C
S29	DA53	C	N3E	SKJ	BA53	K	BZ5F
S30	DA53	D	N4D	SKM	BA53	B	BY4B
S31	DA53	E	N5D	SKP	BA53	H	BZ4F
S32	DA53	F	N6D	SM1	KA51	M	S6D
S33	DA53	H	P1D	SM2	KA51	N	S5D
S34	DA53	J	P2D	SM2(C)SM1	KA54	D	T5B
S35	DA53	K	P3D	SMA	JA54	L	AF3E
S36	DA53	L	P4D	SUR	EA55	L	LE2A

Signal	Signal Location	Pin	Print	Signal	Signal Location	Pin	Print
SWA	EA55	T	LA5A	TW	FA50	F	AC2D
SWB	EA55	S	LA4A	U6	EA49	H	LB2C
SWC	EA55	R	LA3A	UCA	EA49	S	LC5C
SWD	EA55	P	LA2A	UCB	EA49	R	LC4C
SY1	MA52	D	AL3D	UDA	EA49	M	LC1C
$\overline{SY1}$	MA53	D	AL3D	UDB	EA49	L	LB5C
SY2	MA52	C	AL3E	UJE	EA49	A	LA1C
$\overline{SY2}$	MA53	C	AL3E	ULA	EA49	P	LC3C
SYA	MA52	B	AL1D	ULB	EA49	N	LC2C
$\overline{SYA}$	MA53	B	AL1D	UPE	EA49	K	LB4C
SYB	MA52	A	AL1E	URB	EA49	F	LB1C
$\overline{SYB}$	MA53	A	AL1E	URT	EA49	J	LB3C
SZ	BA54	J	AH3B	UWA	EA49	E	LA52
SZ1	BA54	C	AH3E	UWB	EA49	D	LA4C
SZ2	BA54	D	AH4F	UWC	EA49	C	LA3C
TCA	MA49	N	AK5D	UWD	EA49	B	LA2C
TCW *2	MA54	N		WCA	LA50	R	EA3F
$\overline{TCW}\wedge\overline{S(R26)LD}^{*}$	MA54	R	AK5G	WG2	KA51	R	S1A
$\overline{TCW}\wedge\overline{S(R26)LD}$	MA53	T	AK4G	WM	FA50	E	AC2B
TDA	LA49	B	EB5E	WR1	JA54	M	AF4B
$\overline{TDA}$	MA54	L	AK3G	WS1	KA51	C	T2D
TE1	MA49	L	AK5D	WS1(0)AS2	KA54	R	T3E
TE2	MA49	P	AK5E	WS1(C)WS2	KA54	S	T1E
TED	MA49	M	AK5E	WS2	KA51	D	T2F
TKC	BA52	D	BX4D	WS2(C)SM1	KA54	A	T3B
TL1	LA54	A	LH5C	WS2(C)WS1	KA53	P	T2B
TL2	LA55	A	LH4E	ZB	KA52	D	J3F
TR	FA50	D	AC1D	ZE2	KA51	K	S4F
TRF	MA54	S	AK2G				
TSB	MA54	P	AK2G				