Report No. 418

COO-2118-0002

ARITHMETIC UNIT OF ILLIAC III: SIMULATION AND LOGICAL DESIGN- PART II

by

Ping L. Koo
Daniel E. Atkins

Revised by: Lakshmi N. Goyal

October 28, 1968
November 10, 1970

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

COO-2118-0002

Report No. 418

ARITHMETIC UNIT OF ILLIAC III: SIMULATION AND LOGICAL DESIGN- PART II*

by

Ping L. Koo
Daniel E. Atkins

October 28, 1968

Revised & Edited

by

Lakshmi N. Goyal

(November 10, 1970)

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

This report replaces Report No. 290

*Supported in part by Contract AT(11-1)-2118 with the U.S. Atomic Energy Commission.

Preface to the Second Edition

This new issue of the DCS Report No. 290 is an edited version of the old report. There are no significant changes in the new edition except that the minor oversights, omissions and errors have been corrected.

In addition, a few explanatory notes have been added on the flow charts to ease their readability. In case there is any discrepancy between the flow charts and the program listing at the end of the report, it should be resolved in favor of the program.

November 10, 1970

Lakshmi N. Goyal

OUTLINE*

---

*Note: In the body of this report, the leading digits of the section numbers used in this outline may be omitted. The level of the section will be indicated by indentation.

I. INTRODUCTION

1.1 General Description

The arithmetic unit (AU) is being simulated at the hardware level of detail. The simulation is written in PL/1 for the IBM 360/75 computer. It also serves as a dynamic documentation for the logical design of AU.

The AU registers may be simulated either as structure, array or bit string. In the AU, most Boolean operations concerned with any register involves every bit of the register. In this case, bit-string operation is the most suitable operation and less time consuming. Since this simulation program is also to serve as dynamic documentation, it requires a simple, easily understood program which is very close to the actual logical design. The bit-string operation is a very straight forward operation; array or structure operations don't have any particular advantage in this simulation. In case only some bytes or bits of a register are used in the operation, the PL/1 build-in function SUBSTR may be used.

All registers and indicators are simulated as bit strings; all counters are simulated as fixed point number; all gating functions are simulated as procedures and grouped into one subprogram. The Sign-Digit Subtractor (SDS) and propagation logic are simulated as two sub-programs. Every arithmetic order is simulated as one subprogram (including functional sub-blocks, such as quotient selector, multiplier recoder, etc.). This simulator, therefore, is composed of one main program and several subprograms.

Every routine will be explained in detail in the following sections. A summary of the registers, counters and indicators used is given below:

Name	Attributes	Function or Equivalent Component in AU Hardware
M	64 bits	M register
US	64 bits	US register
UM	64 bits	UM register
LS	64 bits	LS register
LM	64 bits	LM register
UQ	65 bits	UQ register
LQ	65 bits	UQ register
UH	65 bits	UH register
LH	65 bits	LH register
V	50 bits	INBUS (5 bytes, 10 bits each)
IV,IV-S	4 bits	Instruction Variants
NT,NT-S	2 bits	Number Type
FA	8 bits	Holds the flags of first operand set to AU
FB	8 bits	Holds the flags of second operand sent to AU
SIGNA	1 bit	Holds the sign of first operand sent to AU.
SIGNB	1 bit	Holds the sign of second operand sent to AU
SR	1 bit	Holds the sign of result format by the AU
EUU	7 bits	Holds the exponent of the first operand sent to AU (Floating pt. number only)
EUM	7 bits	Holds the exponent of the second operand sent to AU
EUL	7 bits	Holds the exponent of the results produced by the AU (floating pt. no. only)

ame	Attributes	Function or Equivalent Component in AU Hardware
	64 bits	The minuend and difference of the signed -digit subtractor (SDS) is represented in SD format. It holds the sign of the minuend. It is used in all 4 stages of SDS.
	64 bits	Holds the magnitude of minuend. Used in all stages of SDS.
	64 bits	Holds subtrahend in conventional binary form. Used in 4 stages of SDS.
	64 bits	Holds the sign of difference. Used in all 4 stages of SDS.
	64 bits	Holds the magnitude of difference. Used in all 4 stages of SDS.
EG	64 bits	A control signal, NEG1 negates the output from SDS. Used in all 4 stages.
	64 bits	Interstage connection for SDS. Used in all 4 stages of SDS.
EGØ	64 bit	NEGØ = 1, complement S into first stage of SDS.
V	1 bit	Overflow indicator
N	1 bit	Underflow indicator
SG	1 bit	Loss of significance indicator
D	1 bit	Invalid data indicator
T	1 bit	'Greater than' indicator
E	1 bit	'Equal' indicator
T	1 bit	'Less than' indicator
F	1 bit	'Flag match' indicator, used in comparison
OGUS	1 bit	'Bogus result' indicator
OLIP	1 bit	PØLIP = 1, for continuous PØLY operation
C	FIX, BIN	Number of division cycles to be performed.

1.2 General Flow

As mentioned in the previous section, the simulator is composed of one main program and several subprograms. The main program is named 'AUSIM', execution starts at AUSIM. A simple pseudo-TP is included to supply operands (or operand), AUSIM calls routines to load these operands to proper registers. After all operands have been loaded upon decoding the Instruction Code, it then calls the routine for that arithmetic order. After completion of that routine, control is switched back to AUSIM for next operation.

Following is a listing of all external procedures defined in this simulator and the local procedures defined in each external procedure. Every external procedure will be explained fully in the following sections together with its associated local procedures.

SUMMARY OF PROCEDURES

External Procedure Name	Entry Name defined	Other Procedures Used	Internal Procedures Defined in this Procedure
AUSIM		CONV, CPRA CVD, CVF, CUL DIV, MPY POLY X, ADDX VDUH1, VDUH2 VDUH3, UDUQ1 UDUQ2, UDUQ3	EADEUU, EBDEUM, FAILFA, FA2RFA FB1LFB, FB2RFB
SDS			
PRØP			
GATE	ASSIM INDFA LHDUH LHL4UH LHL8UH LHR4UH LHR8UH LMDM LMDUM LMDUQ LML8UQ LMR8UQ LQDUQ LQL4UQ	PROP, SDS	

SUMMARY OF PROCEDURES (Continued)

External Procedure Name	Entry Name Defined	Other Procedures Used	Internal Procedures Defined in this Procedure
(GATE)	LQL8UQ		
	LQR4UQ		
	LQR8UQ		
	LSDUH		
	LSDUS		
	LSL8US		
	LSR8US		
	MDY1		
	MDY4		
	ML1Y4		
	ML2Y3		
	ML3Y3		
	ML4Y2		
	ML5Y2		
	ML6Y1		
	ML7Y1		
	NORM		
	PDY4		
	SETBOG		
	T4DLS		
	TIMER		
	UHDLH		
	UHDM		

SUMMARY OF PROCEDURES (Continued)

External Procedure Name	Entry Name Defined	Other Procedures Used	Internal Procedures Defined in this Procedure
(GATE)	UHDUS		
	UHDY1		
	UMDX1		
	UQDLQ		
	UQDUM		
	USDS1		
	VIM1		
	VIM2		
	VDUH1		
	VDUH2		
	VDUH3		
	VDUQ1		
	VDUQ2		
	VDUQ3		
	Z4DIM		
ADDX	SUB	ASSIM	
		LHR4UH, LHRBUH	
	CPRA	IMDUQ, USDS1	
		LQL4UQ, LQL8UQ	
		LQR4UQ, LQR8UQ	
		MDY1, MDY4, NORM	
		PDY4, PROP	
		SETBOG, INDFA	
		SDS, UHDLH	
		UHIM, UMDX1	
		UQDLQ, UQDUM	
		Z4DIM	

SUMMARY OF PROCEDURES (Continued)

External Procedure Name	Entry Name Defined	Other Procedures Used	Internal Procedures Defined in this Procedure
MPY		ASSIM, LHR8UH IMDUM, IMDUQ IMR8UM, LQR8UQ LSDUS, LSR8US MDY1, MDY4 ML1Y4, ML2Y3 ML3Y3, ML4Y2 ML5Y2, ML6Y1 ML7Y1 NORM SDS. T4DLS UHDLH, UHIM UMDX1, UQDLQ USDS1, Z4DIM	MEXTPRE, RECORDER
DIV	CVDDIV	ASSIM, IMIM IMDUM, IMDUQ IML8UM, LQL4UQ LQL8UQ, LQR4UQ LQR8UQ, LSDUS LSL8US, MDY1 MOY4, ML1Y4 ML2Y3, ML3Y3 ML4Y2, ML5Y2 ML6Y1, ML7Y1 NORM, SDS T4DLS	ASSIMQD, ASSIMRE, DOIMD, DIIMD, D2IMD D3IMD, MODDIV DIVID, LDQMS12 LDQMS78, LHL1UH LHL4UH, LHL8UH LHR8UH, IMDUH7 IMDUQ3, IMUQ7, LQL1UQ LQR1UQ, QBDUQH, SELECT

SUMMARY OF PROCEDURES (Continued)

External Procedure Name	Entry Name Defined	Other Procedures Used	Internal Procedures Defined in this Procedure
(DIV)		UHDLH, UHIM	TENDY4, TENL1Y4
		UHDUS, UMDX1	TENL2Y3, TENL3Y3
		UQDLQ, UQDUM	TENL4Y2, TENL5Y2
		Z4DIM	TENL6Y1, TENL7Y1
CONVS	CVL	ASSIM, CVDDIV	DVB
	CVF	LHL4UH, LHL8UQ	
	CVD	LMIM, LMDUM	
		LMDUQ, LQL4UQ	
		LQL8UQ, LQR4UQ	
		LQR8UQ, LSDUH	
		LSDUS, MDY4	
		ML1Y4, ML3Y4	
		ML4Y2, SDS	
		T4DLS, UHDLH	
		UHDUS, UMDX1	
		UQDLQ, UQDUM	
		USDS1, Z4DIM	

II. Pseudo-Simulation of TP and Operand Gating

1. Name: AUSIM

2. Functional Description:

This is a main program, this program includes two sections:

(1) A simple pseudo-simulation of TP, which sends to the AU via the INBUS, a control byte containing the instruction variant (IV), the number type (NT) and the operands. The operand will be sent, one word at a time, in the following order: left word of operand A, left word of operand B, then the right word of operand A, and finally the right word of operand. Each byte of the word consists of 10 bits: 8 data bit, 1 flag bit and 1 parity bit.

(2) Upon rapid initial decoding of the instruction variant and number type, AUSIM loads the operands as received, one word at a time, into appropriate registers. After all operands have been loaded the control of execution is switched to an appropriate subprogram according to the instruction variant. After the execution of that subprogram is completed, control is switched back to AUSIM for the next arithmetic order.

3. Formal Calling Sequence:

Not applicable to a main program

4. Formal Parameter Description: None

5. Implicit Parameter Description:

Initially, all registers, counters and indicators will be cleared. Depending upon the instruction variant and number type- part or all of UH, UQ, FA, FB, EUU, and EUM registers will be loaded from INBUS.

6. Subroutines Used:

ADDX, SUB, CPRA, MPY, DIV, CVL, CVF, CVD, POLYX, VDUH1, UDUH2, UDUH3, VDUQ1, UDUQ2, VDUQ3

7. Operational Description:

7.1 English Text:

The number contained in small square associated with some blocks in the flowchart will be used in the following paragraphs for easy references.

(1) One parameter, PRINT, is used to control the print out of the intermediate result. If PRINT = 1, both inputs and outputs of signed-digit subtractor will be printed out (it is useful for debugging), otherwise printing in SDS will be skipped. Start from block [1]: all registers have been cleared. Since the indicator 'POLIP' is set to 1 after the first operation of POLY, and will be reset to 0 after the last operation of POLY, the indicator 'POLIP' will not be cleared at this point, if the arithmetic order is POLY.

(2) In the Illiac III hardware, operands are retrieved from Operand Stack (ØS). At present status, the AU has been tested using a pseudo-simulated TP. The operands are taken from external input device. Blocks [2] to [6] are used to read operands (or operand) from an input device (cards) and convert them into internal representation as if they were being retrieved from ØS. (64-bit storage area is used. For operand less than 64 bits, it is left adjusted in the area). The number of operands needed is determined from the Instruction Variant (IV). For data conversion type orders (first bit of IV equals 0), only one operand is needed (NOP = 1) otherwise two operands are needed (NOP = 2). The only exception is POLY: for the first time, two operands are needed, but for subsequent POLY, only one operand is needed.

The order of input data will be: instruction variant, number type, IND, operand A, flag bits of operand A, operand B, flag bits of operand B. Operand may be in either conventional form (IND = 0) or in their internal bit-form (IND = 1).

(3) The order of operands sent from TP to AU is: left word of operand A left word of operand B, right word of operand A then right word of operand B.

Block [6] is used to re-arrange the operands and their associated flag bits in this order and place them in temporary buffer for later usage.

(4) Starting at block [7], operands kept in the temporary buffer will be sent to the AU, one word at a time, via the INBUS. The word carried by INBUS has 5 bytes, 10 bits each. In this simulation parity bits are neither set nor checked.

INBUS WORD		Corresponding Bits of Operand Word
Byte No.	Bit No.	
1	4-7	IV
	8-9	NT
2	11-18	Data bits of 1st byte.
	19	Flag bit of 1st byte
3	21-28	Data bits of 2nd byte
	29	Flag bit of 2nd byte
4	31-38	Data bits of 3rd byte
	39	Flag bit of 3rd byte
5	41-48	Data bits of 4th byte
	49	Flag bit of 4th byte

TABLE 2-1: Format of INBUS Word

For floating point numbers and decimal numbers, each operand consists of 1 double (WRD = 2), for fixed point number each operand consists of 1 word (WRD = 1); therefore, the total number of words to be transmitted = WRD x NOP.

According to the instruction variant and number type, the word transmitted via INBUS will be loaded into the proper positions of the appropriate registers by the routines defined in external procedure GATE. Every word in the temporary will be sent in a similar manner until all words have been loaded. The following is a table summarizing the contents of registers after the loading of operands is completed.

TABLE 2-2: Summary of Loading Operands into Registers

Combination of NT and IV	Subprogram Used	Contents of Registers After Loading
FX • POLY • POLIP	VDUH2	(UH (bytes 4-7) = New Coeff. Sign B = Sign of new coeff. EUM = Exponent bits of coeff. UH (bytes 1-7) = Fraction bit of coeff.
FL•POLY•POLIP	VDUH1, VDUH2 EBDEUM	
FL (POLY•$\overline{\text{POLIP}}$	VDUQ1, VDUQ2	SIGNA = Sign of Operand A, SIGNB = Sign of Operand B, FA = Flag bits of Operand A FB = Flag bits of Operand B EUU = Exponent bits of Operand A EUM = Exponent bits of Operand B UQ (bytes 1-7) = Fraction of Op. A UH (bytes 1-7) = Fraction of Op. B
ADDvSUBvCPRA	VDUH1, VDUH2,	
MPYvDIv)	FAILFA, FA2RFA	
	E ADEUU, EBDEUM	
	FB1LFB, FB2RFB	
X•MPY	FAILFA, FB1FB	SIGNA = Sign of multiplicand SIGNB = Sign of multiplier FA_{1-4} = Flag bits of multiplicand FB_{1-4} = Flag bits of multiplier UQ (Bytes 4-7) = Multiplicand UH (Bytes 0-3) = Multiplier
	VDUQ2, VDUH3	

TABLE 2-2: Summary of Loading Operands into Registers (Continued)

Combination of NT and IV	Subprogram Used	Contents of Registers After Loading
FX•DIV	FAILFA, FB1LFB VDUQ3, VDUQ2 VDUH3	SIGNA = Sign of dividend SIGNB = Sign of divisor FA_{1-4} = Flag bits of dividend FB_{1-4} = Flag bits of divisor UQ(Bytes 0-3) = Dividend UQ (Bytes 4-7) = UH (Bytes 0-3)= Divisor
FX•CVD	VDUQ2 FAILFA	SIGNA = Sign of Operand A FA_{1-4} = Flag bits of Operand A UQ (Bytes 4-7) = Operand A
FX•CVF	VDEQ3 FAILFA	SIGNA = Sign of Operand A FA_{1-4} = Flag bits of Operand A UQ (Bytes 0-3) = Operand A
FL•(CVDvCVL)	VDUQ1, VDUQ2 FAILFA, FA2RFA EBDEUM	SIGNA = Sign of Operand A FA = Flag of Operand A EUM = Exponent bits of Operand A UQ(Bytes 1-7) = Fraction bits of Operand A
DEC•(CVLvCVF)	VDUQ1, VDUQ2 FAILFA, FA2RFA	SIGNA = Sign of Operand A FA = Flag bits of Operand A

(5) After all operands have been loaded (according to the instruction variant) one of the external procedures of the arithmetic order is chosen, and control of execution is switched to that procedure.

7.2 Flowchart:

Flowchart is attached

The symbols used in the flowchart are:

POLIP: Indicator, is to be set to 1 for subsequent POLY

IV: Instruction variant (4 bit)

NT: Number Type (2 bit)

NOP: Number of operands

IND: = 0 if the input operands are in conventional fixed, floating or decimal form.

= 1 if the input operand are in internal bit representations.

STRING1: Internal representation of Operand A (64 bit)

OPF1: Flag bits of Operand A (8-bit)

STRING2: Internal representation of Operand B (64-bit)

OPF2: Flag bits of Operand B (8-bit)

TEMP(I): I = 1, 2, 3, 4; Temporary buffer for data bits operands re-arranged in the order for transmission. (32-bit each).

FLAG(I): I = 1, 2, 3, 4: Temporary buffer for the flag bits associated with data bits in TEMP(I). (4 bits each)

V: INBUS word (50 bits)

WRDCT: A counter to count number of words being transmitted

SIGNA: Sign register for Operand A.

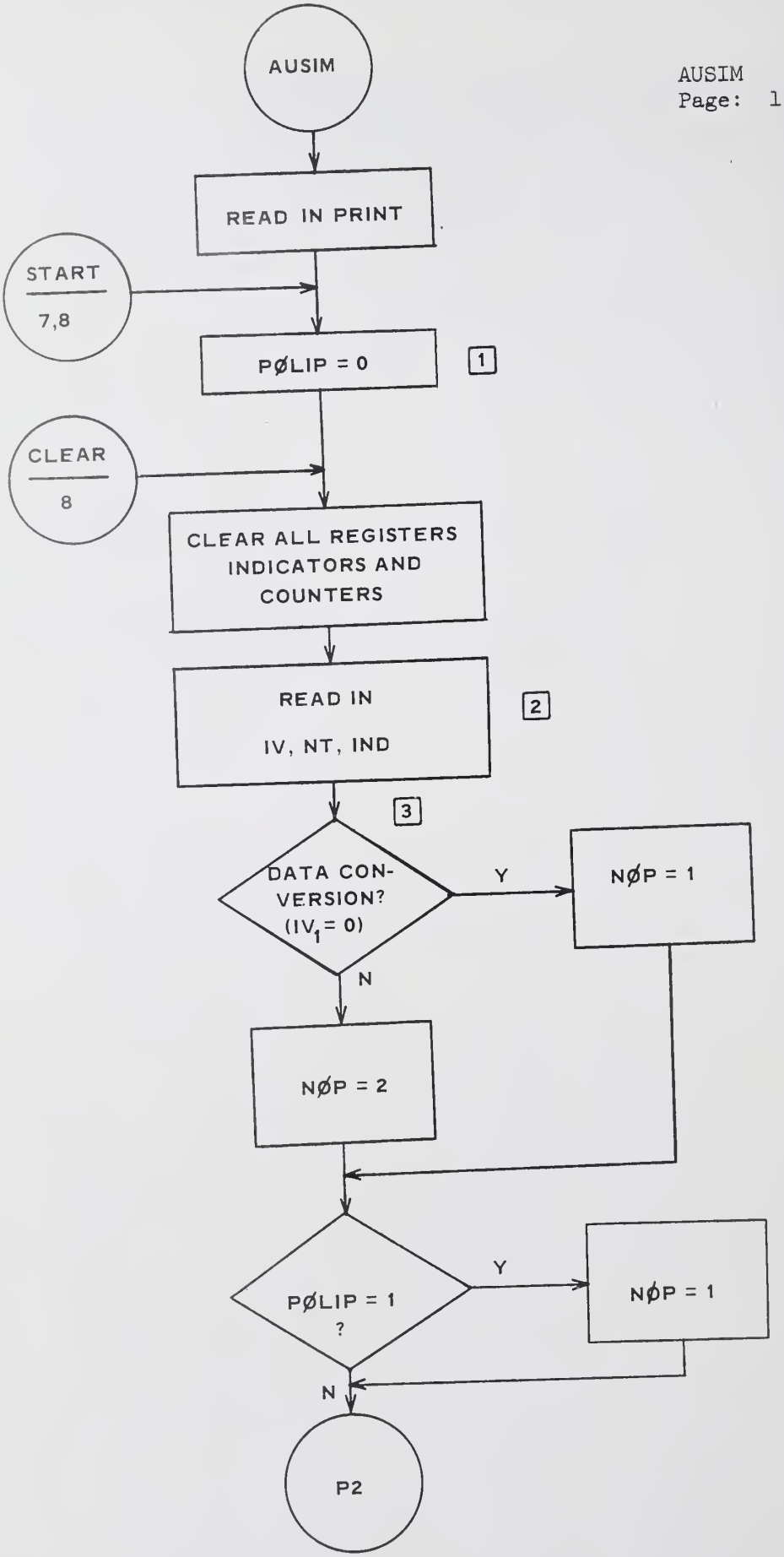
SIGNB: Sign register for Operand B

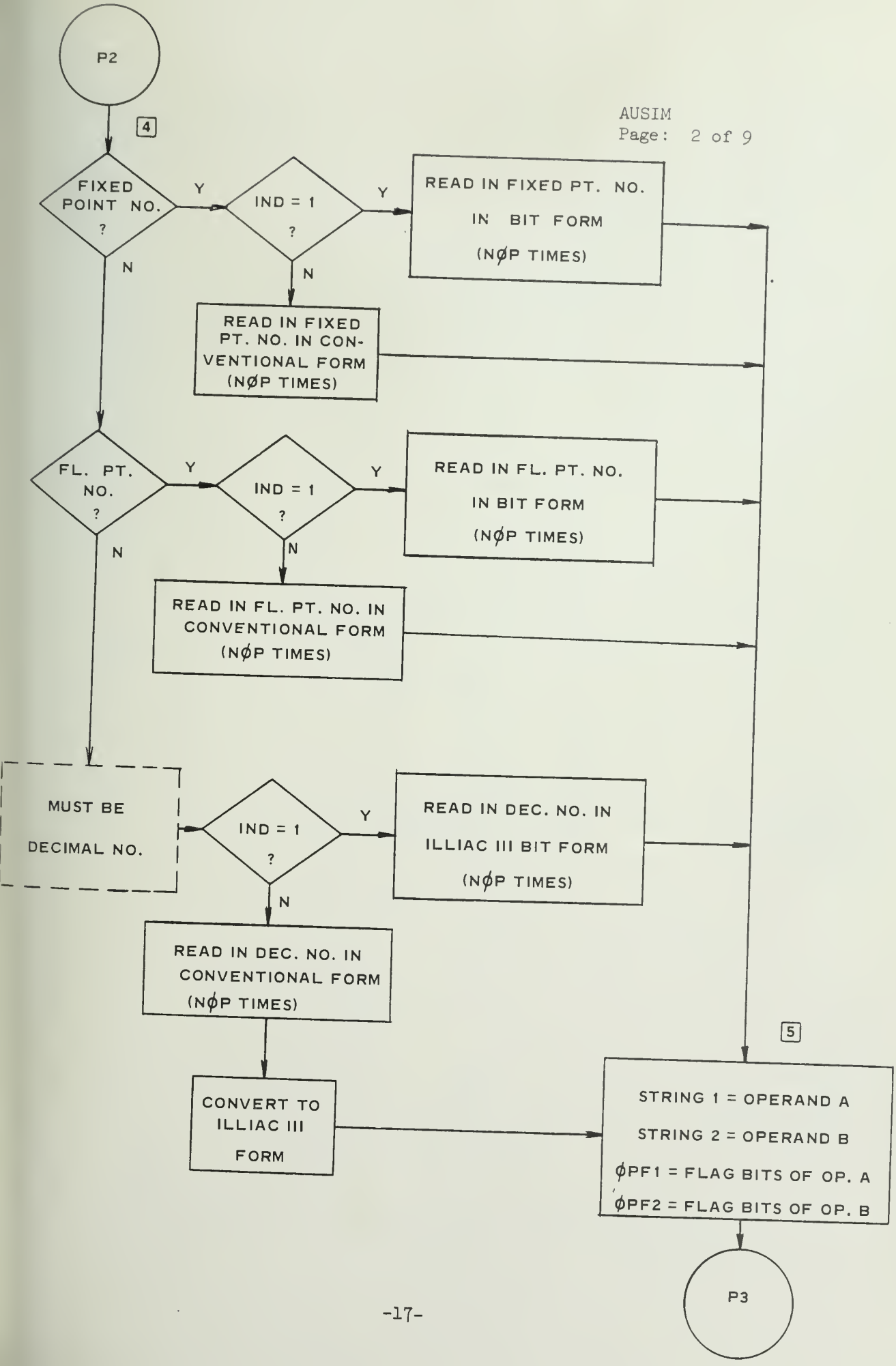
FA: Flag register for Operand A

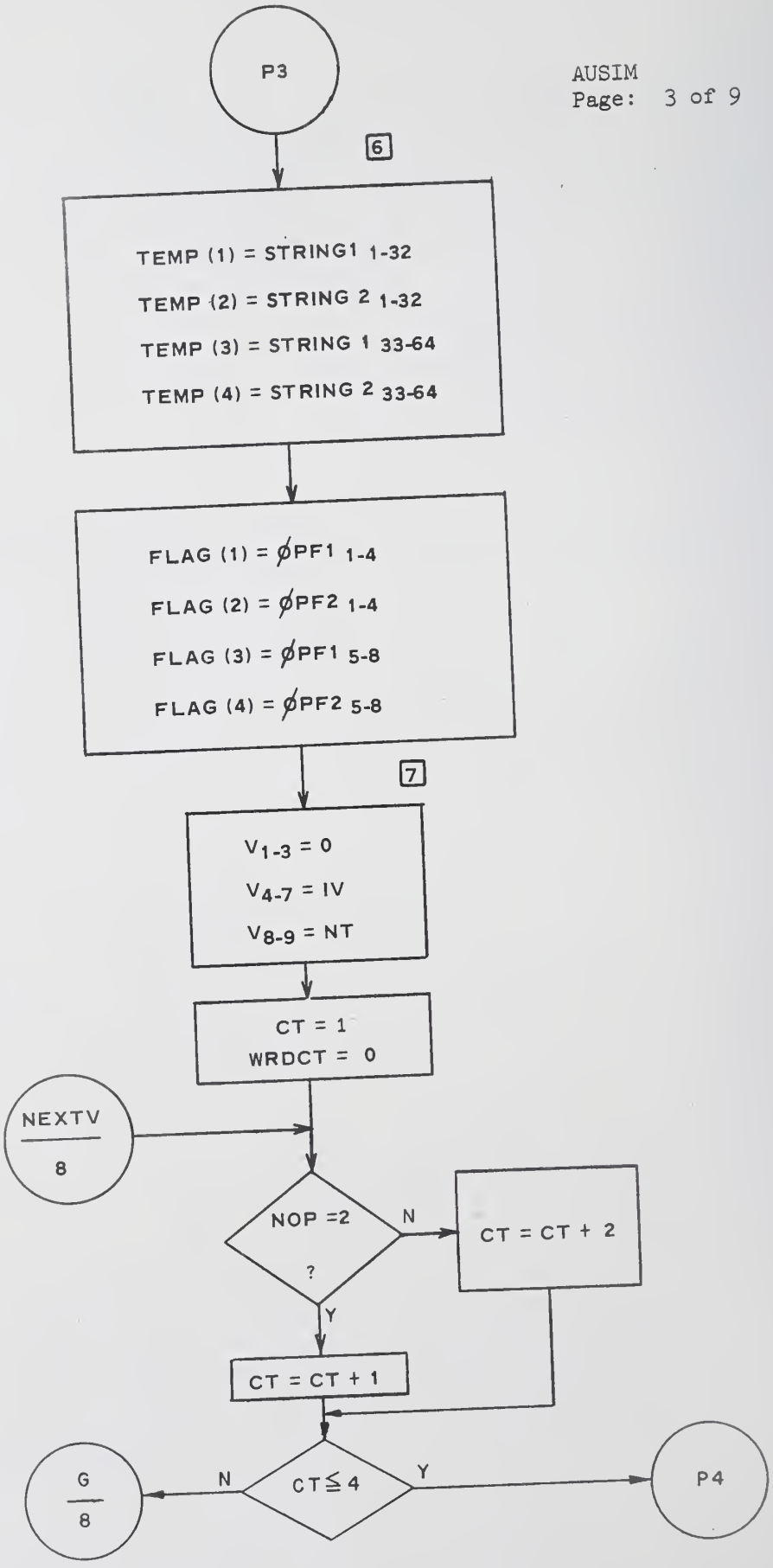
FB: Flag register for Operand B.

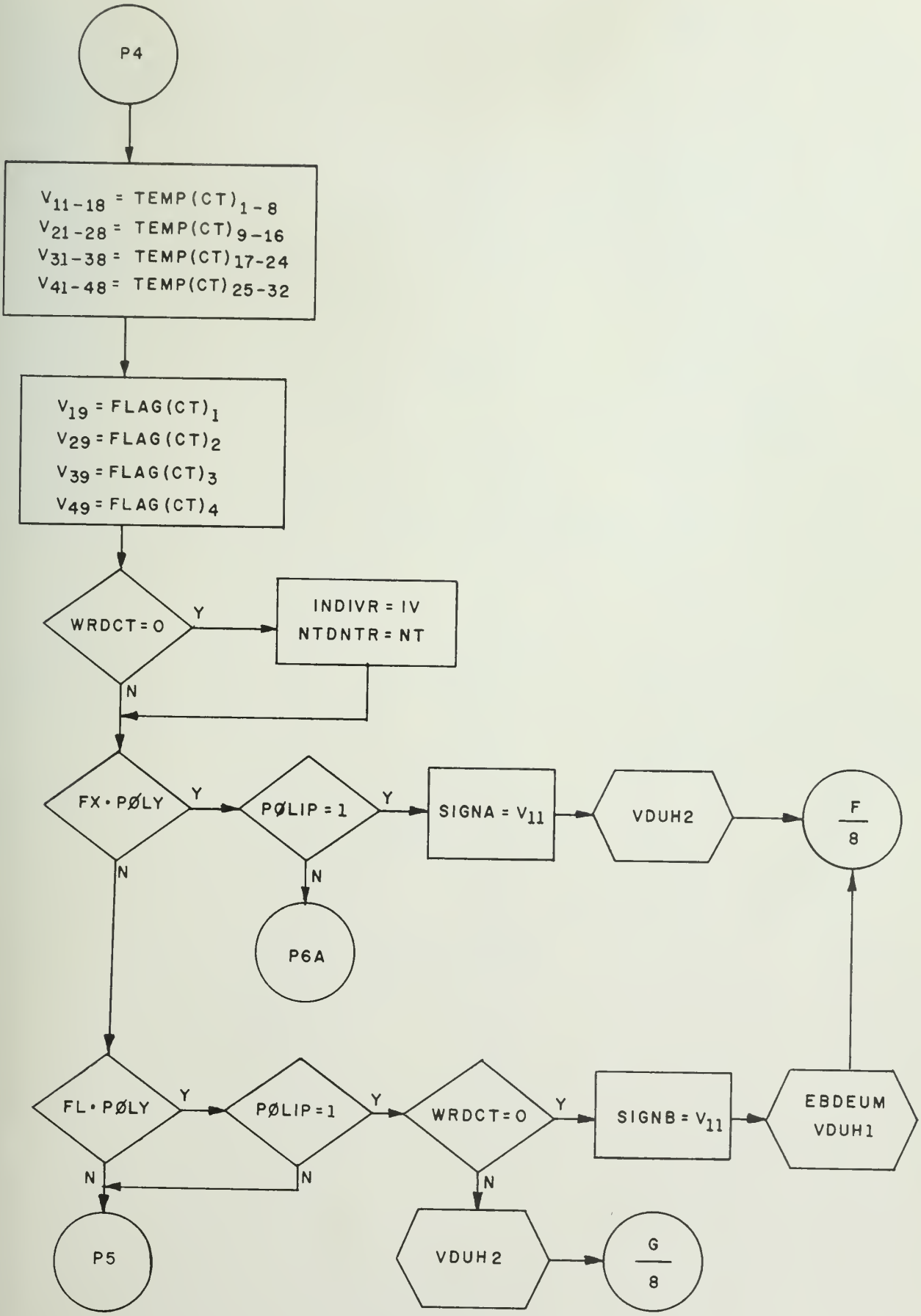
EUU: Exponent register for Operand A.

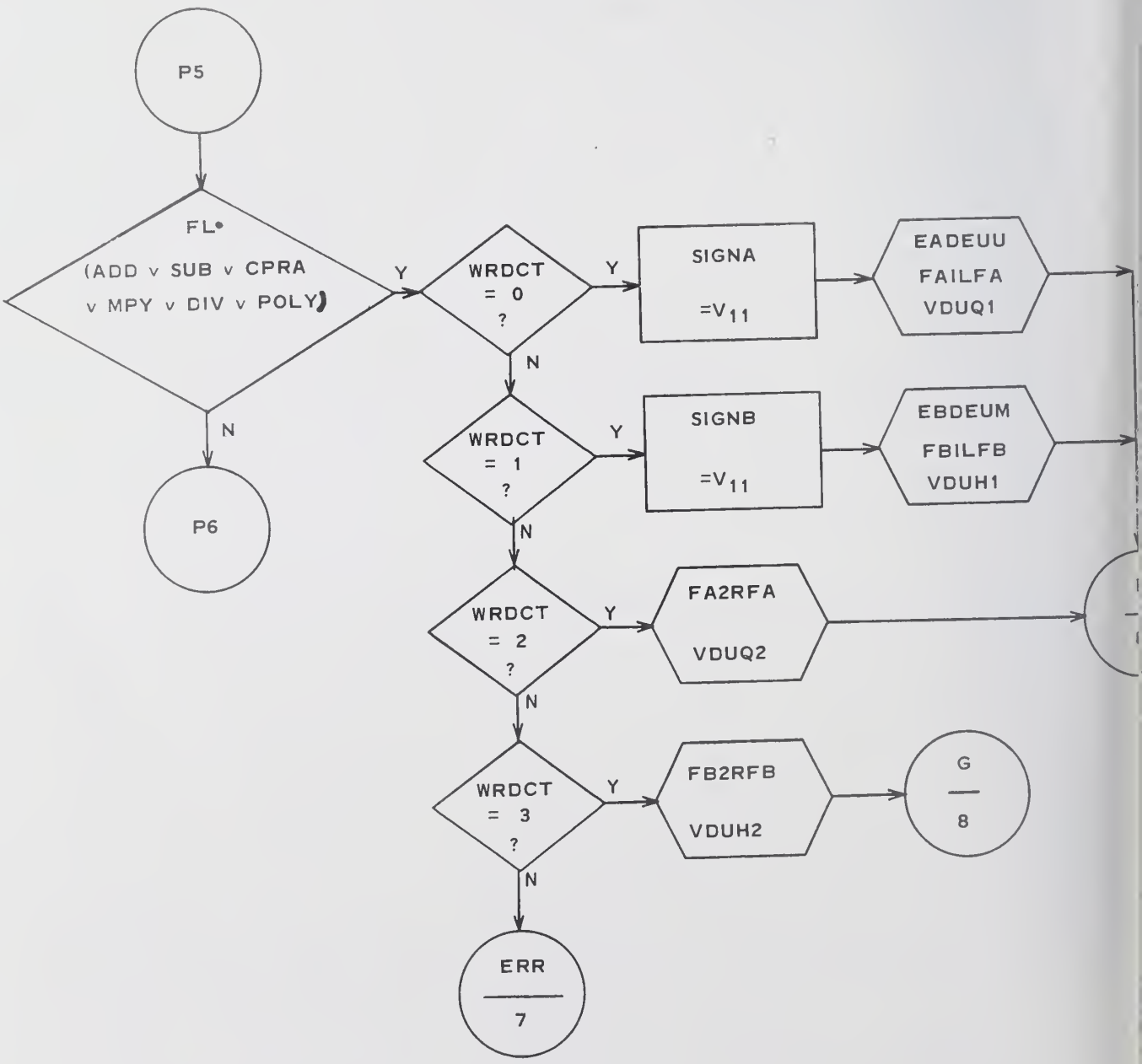
EUM: Exponent register for Operand B.

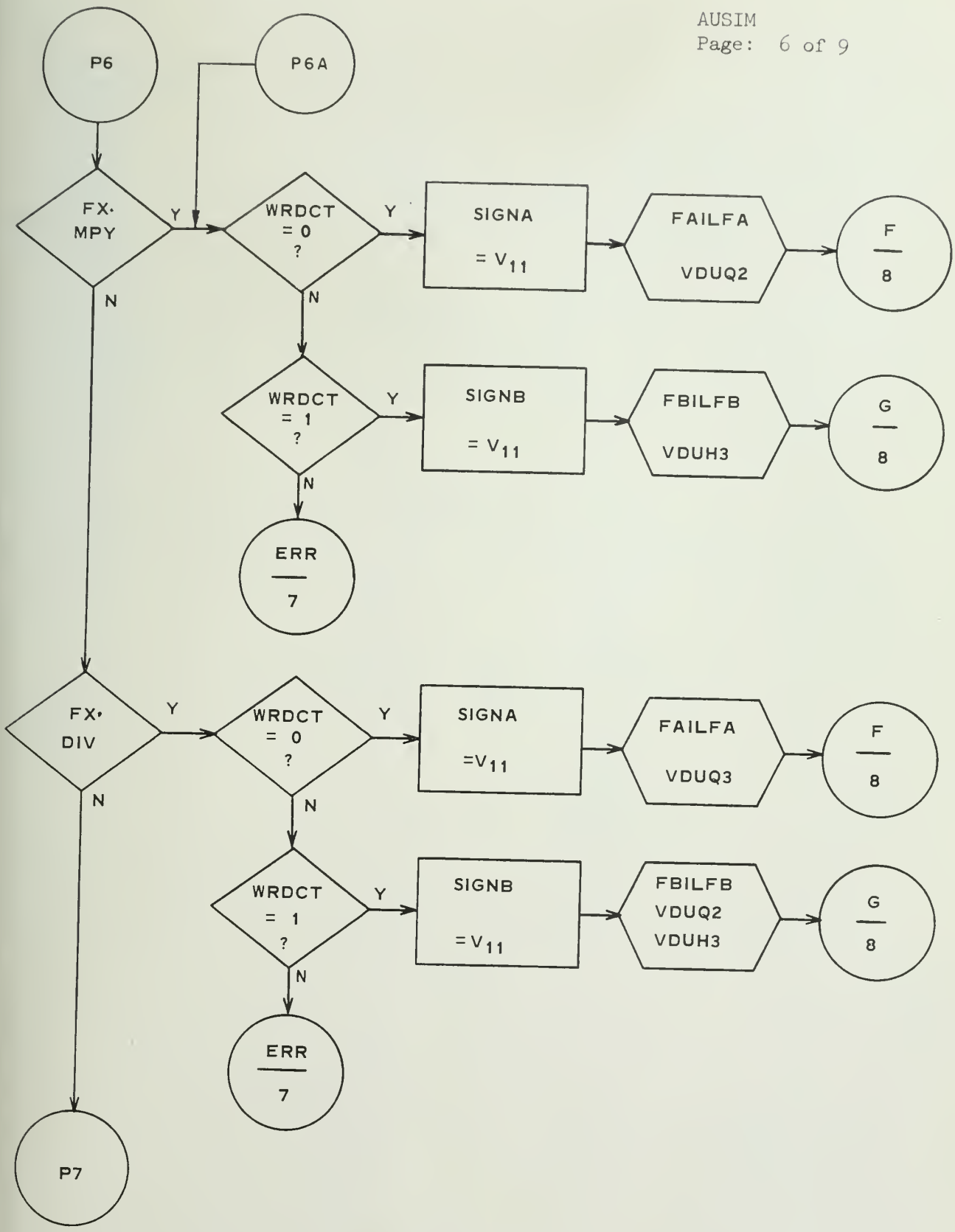

AUSIM
READ IN PRINT
START
7,8
PØLIP = 0
1
CLEAR
8
CLEAR ALL REGISTERS INDICATORS AND COUNTERS
READ IN
IV, NT, IND
2
3
DATA CON-VERSION? (IV_1 = 0)
Y
NØP = 1
N
NØP = 2
PØLIP = 1 ?
Y
NØP = 1
N
P2

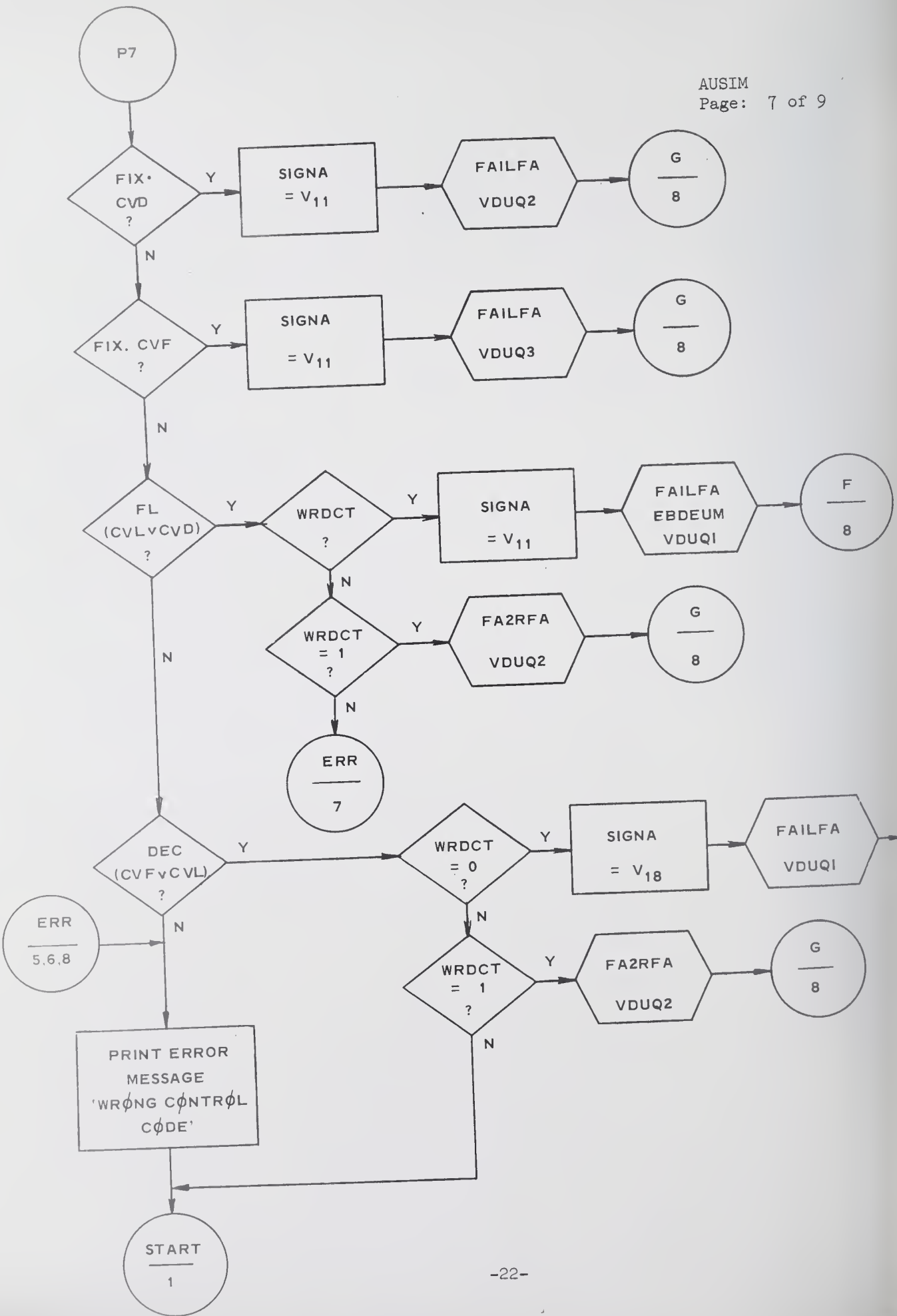

P2
4
FIXED POINT NO. ?
Y
IND = 1 ?
Y
READ IN FIXED PT. NO. IN BIT FORM (NØP TIMES)
N
N
READ IN FIXED PT. NO. IN CONVENTIONAL FORM (NØP TIMES)
FL. PT. NO. ?
Y
IND = 1 ?
Y
READ IN FL. PT. NO. IN BIT FORM (NØP TIMES)
N
N
READ IN FL. PT. NO. IN CONVENTIONAL FORM (NØP TIMES)
MUST BE DECIMAL NO.
IND = 1 ?
Y
READ IN DEC. NO. IN ILLIAC III BIT FORM (NØP TIMES)
N
READ IN DEC. NO. IN CONVENTIONAL FORM (NØP TIMES)
CONVERT TO ILLIAC III FORM
5
STRING 1 = OPERAND A
STRING 2 = OPERAND B
ØPF1 = FLAG BITS OF OP. A
ØPF2 = FLAG BITS OF OP. B
P3

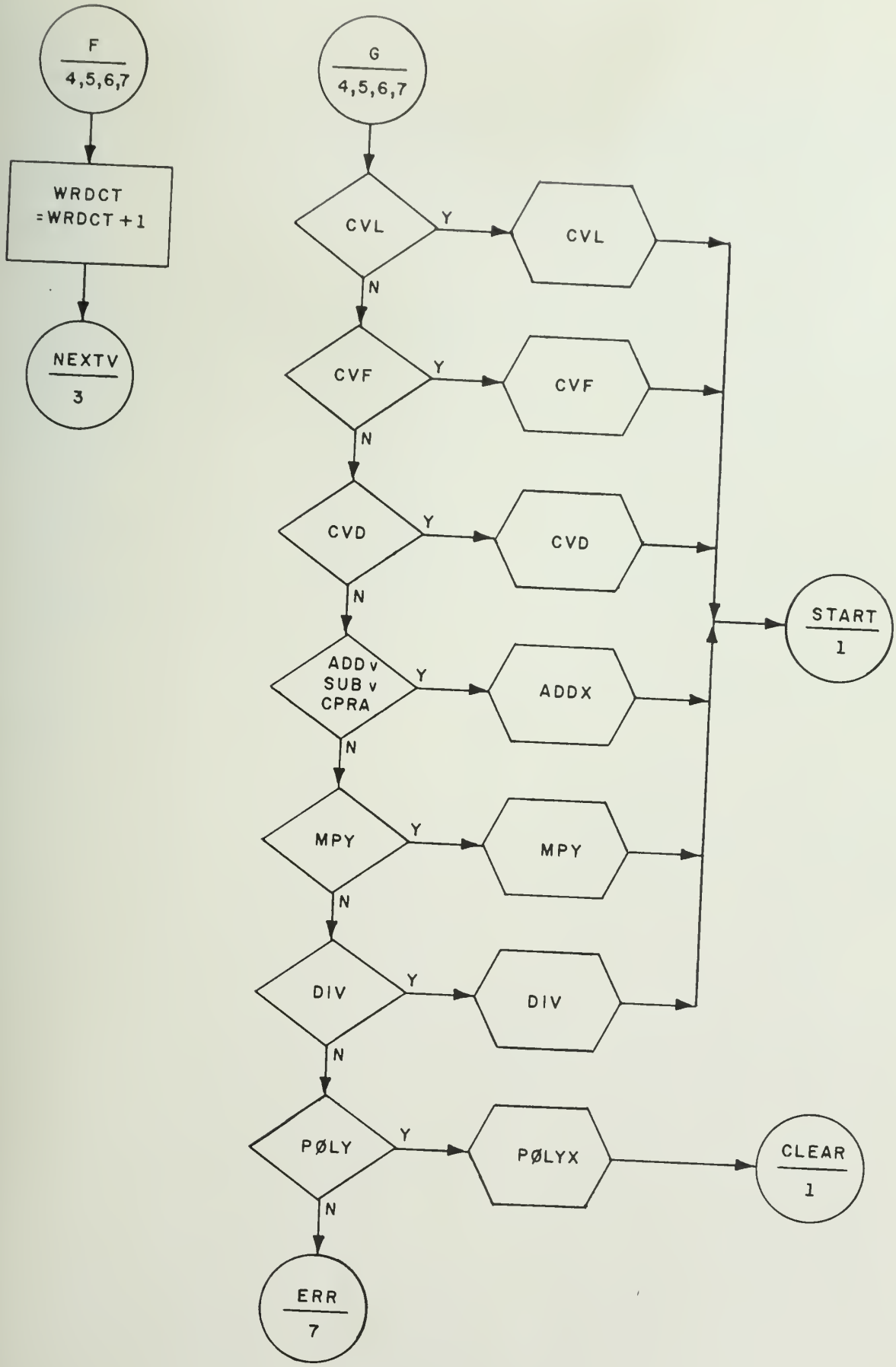

P3
6
TEMP (1) = STRING1 1-32
TEMP (2) = STRING 2 1-32
TEMP (3) = STRING 1 33-64
TEMP (4) = STRING 2 33-64
FLAG (1) = ØPF1 1-4
FLAG (2) = ØPF2 1-4
FLAG (3) = ØPF1 5-8
FLAG (4) = ØPF2 5-8
7
V1-3 = 0
V4-7 = IV
V8-9 = NT
CT = 1
WRDCT = 0
NEXTV
8
NOP =2
?
N
CT = CT + 2
Y
CT = CT + 1
G
8
N
CT≦4
Y
P4

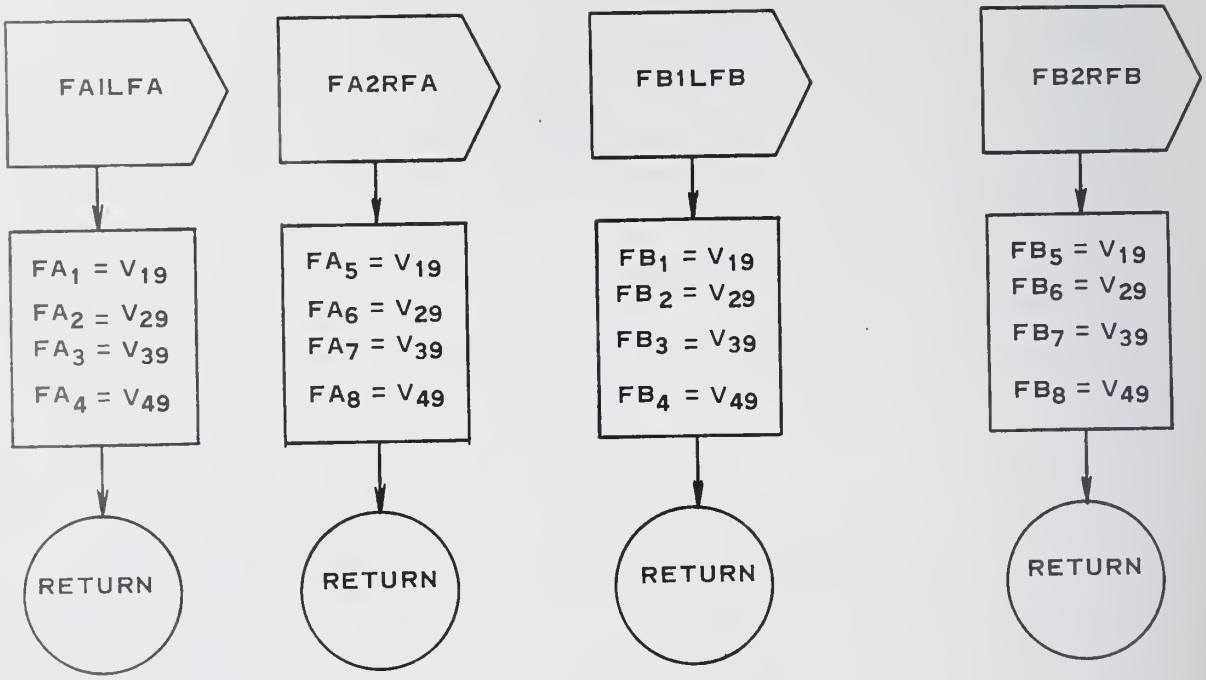

P4
$V_{11-18} = TEMP(CT)_{1-8}$
$V_{21-28} = TEMP(CT)_{9-16}$
$V_{31-38} = TEMP(CT)_{17-24}$
$V_{41-48} = TEMP(CT)_{25-32}$
$V_{19} = FLAG(CT)_1$
$V_{29} = FLAG(CT)_2$
$V_{39} = FLAG(CT)_3$
$V_{49} = FLAG(CT)_4$
WRDCT = 0
Y
INDIVR = IV
NTDNTR = NT
N
FX • PØLY
Y
PØLIP = 1
Y
SIGNA = V_{11}
VDUH2
F / 8
N
N
P6A
FL • PØLY
Y
PØLIP = 1
Y
WRDCT = 0
Y
SIGNB = V_{11}
EBDEUM
VDUH1
N
N
N
P5
VDUH2
G / 8

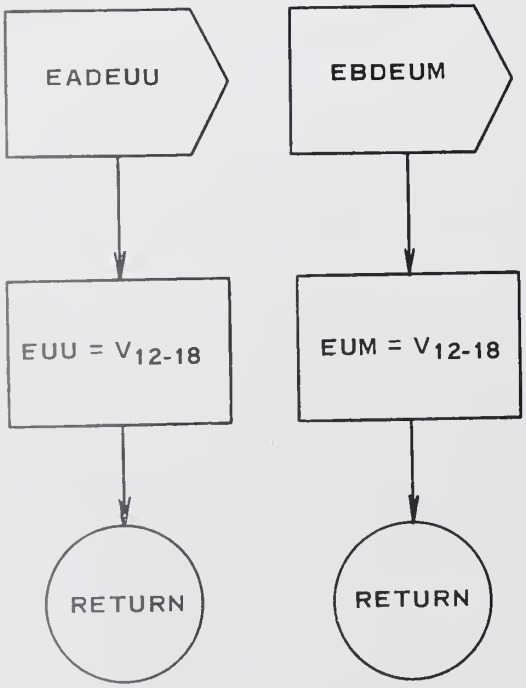

P5
FL•
(ADD v SUB v CPRA
v MPY v DIV v POLY)
Y
N
P6
WRDCT
= 0
?
Y
N
SIGNA
=V11
EADEUU
FAILFA
VDUQ1
WRDCT
= 1
?
Y
N
SIGNB
=V11
EBDEUM
FBILFB
VDUH1
WRDCT
= 2
?
Y
N
FA2RFA
VDUQ2
WRDCT
= 3
?
Y
N
FB2RFB
VDUH2
G
8
ERR
7


P6
P6A
FX·
MPY
Y
N
WRDCT
= 0
?
Y
N
SIGNA
= V11
FAILFA
VDUQ2
F
8
WRDCT
= 1
?
Y
N
SIGNB
= V11
FBILFB
VDUH3
G
8
ERR
7
FX·
DIV
Y
N
WRDCT
= 0
?
Y
N
SIGNA
=V11
FAILFA
VDUQ3
F
8
WRDCT
= 1
?
Y
N
SIGNB
=V11
FBILFB
VDUQ2
VDUH3
G
8
ERR
7
P7


P7
FIX• CVD ?
Y
SIGNA = V11
FAILFA VDUQ2
G / 8
N
FIX. CVF ?
Y
SIGNA = V11
FAILFA VDUQ3
G / 8
N
FL (CVL∨CVD) ?
Y
WRDCT ?
Y
SIGNA = V11
FAILFA EBDEUM VDUQI
F / 8
N
WRDCT = 1 ?
Y
FA2RFA VDUQ2
G / 8
N
ERR / 7
N
DEC (CVF∨CVL) ?
Y
WRDCT = 0 ?
Y
SIGNA = V18
FAILFA VDUQI
N
WRDCT = 1 ?
Y
FA2RFA VDUQ2
G / 8
N
ERR / 5,6,8
N
PRINT ERROR MESSAGE 'WRØNG CØNTRØL CØDE'
START / 1


F
4,5,6,7
WRDCT
=WRDCT+1
NEXTV
3
G
4,5,6,7
CVL
Y
CVL
N
CVF
Y
CVF
N
CVD
Y
CVD
N
ADD v
SUB v
CPRA
Y
ADDX
N
MPY
Y
MPY
N
DIV
Y
DIV
N
PØLY
Y
PØLYX
N
ERR
7
START
1
CLEAR
1


FAILFA
FA1 = V19
FA2 = V29
FA3 = V39
FA4 = V49
RETURN
FA2RFA
FA5 = V19
FA6 = V29
FA7 = V39
FA8 = V49
RETURN
FB1LFB
FB1 = V19
FB2 = V29
FB3 = V39
FB4 = V49
RETURN
FB2RFB
FB5 = V19
FB6 = V29
FB7 = V39
FB8 = V49
RETURN


EADEUU
EUU = V12-18
RETURN
EBDEUM
EUM = V12-18
RETURN

8. Error Condition:

When invalid instruction variant is used, an error message will be printed out, and execution for that set of input data will be terminated. Control of execution is switched to the beginning of AUSIM for the next set of data.

9. Local Procedures:

9.1 FAILFA: This routine is used to load bit 19, 29, 39, 49 into the left half of flag register FA.

9.2 FA2RFA: This routine is used to load bit 19, 29, 39, 49 into the right half of flag register FA.

9.3 FB1LFB: Same as FAILFA; FB is used instead of FA.

9.4 FB2RFB: Same as FA2RFA; FB is used instead of FA.

9.5 EADEUU: To load bit 12 through bit 18 of V into the exponent register EUU.

9.6 EBDEUM: To load bit 12 through bit 18 of V into the exponent register EUM.

III: Gate Functions and Shifting Logic

All gate functions, shifting logics, assimilation of output of signed-digit subtractor and some other miscellaneous functional subblocks are grouped into one procedure 'GATE' with different entry names. Each of them will be fully described below. One overall flowchart will be provided.

1. Operand Loading Gate Functions:

VIM1: Load from INBUS (V) byte 3-5 (bit 1-8) into M register byte 1-3.

VIM2: Load from INBUS (V) byte 2-4 (bits 1-8 of each byte) into M register byte 4-7.

VDUQ1: Load from INBUS (V) byte 2-4 (bits 1-8 of each byte) into UQ register byte 1-3.

VDUH1: Same as VDUQ1, UH register is loaded instead of UQ.

VDUQ2: Load from INBUS (V) byte 1-4 (bit 1-8 of each byte) into UQ register byte 4-7.

VDUH2: Same as VDUQ2, UH register is loaded instead of UQ.

VDUQ3: Load from INBUS (V) byte 1-4 (bit 1-8 of each byte) into UQ register byte 0-3.

VDUH3: Same as VDUQ3, UH register is loaded instead of UQ

2. M-Register Shifting Gate Functions

MDY1: Contents of M register is used as Y-input to SDS (Stage 1).

MDY4: Contents of M register is used as Y-input to SDS (Stage 4)

ML7Y1: Contents of M register is left shifted 7 bits and is used as Y-input to SDS (Stage 1).

ML6Y1: Contents of M register is left shifted 6 bits and is used as Y-input to SDS (Stage 1).

ML5Y2: Contents of M register is left shifted 5 bits and is used as Y-input to SDS (Stage 2)

ML4Y2: Contents of M register is left shifted 4 bits and is used as Y-input to SDS (Stage 2).

ML3Y3: Contents of M register is left shifted 3 bits and is used as Y-input to SDS (Stage 2).

ML3Y3: Contents of M register is left shifted 3 bits and is used as Y-input to SDS (Stage 3).

ML2Y3: Contents of M register is left shifted 2 bits and is used as Y-input to SDS (Stage 3).

ML1Y4: Contents of M register is left-shifted 1 bit and is used as Y-input to SDS (Stage 4).

3. General Gate Functions:

UHDY1: Contents of UH register is used as Y-input to SDS (Stage 1)

PDY4: The borrow/carrier bits generated by the propagation logic is used as Y-input to SDS (Stage 4).

LMDM: Contents of LM register is directly transmitted to M register.

LML8UM: Contents of LM register left shifted 8 bits and transmitted to UM register.

LMR8UM: Contents of LM register is right shifted 8 bits and transmitted to UM register.

LMDUM Contents of LM register is transmitted to UM register.

LMDUQ Contents of LM register is transmitted to UQ register.

LHL4UH: Contents of LH register is left shifted 4 bits and transmitted to UH register.

LHR4UH: Contents of LH register is right shifted 4 bits and transmitted to UH register.

LHL8UH: Contents of LH register is left shifted 8 bits and transmitted to UH register.

LHR8UH: Contents of LH register is right shifted 8 bits and transmitted to UH register.

LHDUH: Contents of LH register is transmitted to UH register.

LQL4UQ: Contents of LQ register is left shifted 4 bits and transmitted to UQ register.

LQR4UQ: Contents of LQ register is right shifted 4 bits and transmitted to UQ register.

LQL8UQ: Contents of LQ register is left shifted 8 bits and transmitted to UQ register.

LQR8UQ: Contents of UQ register is right shifted 8 bits and transmitted to UQ register.

LQDUQ: Contents of LQ register is transmitted to UQ register.

LSL8US: Contents of LS register is left shifted 8 bits and transmitted to US register.

LSR8US: Contents of LS register is right shifted 8 bits and transmitted to US register.

LSDUH: Contents of LS register is transmitted to UH register.

LSDUS: Contents of LS register is transmitted to US register.

UHDM: Contents of UH register is transmitted to M register.

UHDUS: Contents of UH register is transmitted to US register.

UHDLH: Contents of UH register is transmitted to LH register.

UMDX1: Contents of UM register is used as X-input to SDS (Stage 1).

UQDUM: Contents of UQ register is transmitted to UM register.

UQDLQ: Contents of UQ register is transmitted to LQ register

USDS1: Contents of US register is used as S-input to SDS (Stage 1).

T4DLS: The T-output of SDS (Stage 4) is transmitted to LS register.

Z4DLM: The Z-output of SDS (Stage 4) is transmitted to LM register.

4. Assimilation Logic

1. Name: ASSIM

2. Functional Description:

The result of signed-digit subtractor is in the SDS format, this routine is used to convert it into the conventional binary form, and the assimilated result will be Z (at Stage 4).

3. Formal Calling Sequence:

CALL ASSIM

4. Formal Parameter Description: None

5. Implicit Parameter Description:

The proper values for the bit-strings X, Y, S, N of SDS (Stage 1) must be set before this routine is called.

6. Subroutines Used:

SDS, PRØP

7. Operational Description

7.1 English Text:

To convert data in the SDS format into the conventional binary form requires borrow propagation and one additional subtraction.

(1) The input X, S, Y to SDS (Stage 1) and the negation control signal N is passed to this routine by the calling program. The SDS will perform an addition or subtraction according to N.

(2) The borrow/carry bits are generated by the routine 'PRØP'.

(3) The output T, Z, of SDS (Stage 1) will pass through SDS (Stage 2 and 3) by simply setting Y = N = G = 0 and X = Z, S = T.

(4) Finally, the borrow/carry bits is used as the Y-input of SDS (Stage 4). The assimilated result is formed at Z.

7.2 Flowchart

Flowchart is attached.

(5) Normalization

1. Name: NØRM

2. Functional Description:

For floating point ADD, SUB, MPY and DIV, normalization of the final result is usually needed. If overflow or underflow occurs during the normalization process, the bogus result indicator will be setted and FA register

will be set accordingly. In this case FA is used to hold error indicators and return them to TP.

3. Formal Calling Sequence

   CALL NØRM

4. Formal Parameter Description: None

5. Implicit Parameter Description:

The UQ register holds the fraction portion of the floating number to be normalized, the exponent portion is held in the temporary storage II (this location is used by all subporgrams in this simulator).

6. Subroutines Used:

   UQDLQ, LQL4UQ, LQR4UQ, SETBØG, LQL8UQ

7.1 Operational Description:

7.1.1 UQ register holds the fraction portion. If bit 5 through bit 8 all not zeroes, the UQ register must right shift 4 bits and the exponent will be increased by 1. (In multiplication, bits 1-4 is always zero due to a right shift to make space available for the exponent in the ultimate result to be transmitted.) If the final exponent is greater than 63 (in AU hardware, a 7-bit adder **is used**,but in the simulation, the best available facility is a half word addition, therefore II greater than 127 is tested) overflow occurs. The BØGUS and ØV indicator will be set and EUL set to all 1's.

7.1.2 If bit 9 through bit 12 are all zeroes, the UQ register is left shifted until UQ_{9-12} are not all zeroes. If UQ_{9-16} are all zeroes, UQ is left shifted 8 bits, and the exponent is reduced by 2. If only UQ_{9-13} are all zeroes, UQ is left shifted 4 bits and the exponent is decreased by 1. In case of underflow (exponent less than -64). BOGUS and UN indicators will be set and EUL set to all 0's.

7.2 Flowchart

Flowchart is attached.

(6) Set Bogus result and error indicators

SETBØG: If ØV = 1 then EUL (exponent of result) is set to all 1's; if UN = 1 then EUL is set to all 0's. If any one of the 4 indicators: ØV, UN, LSG, ID is on then the BØGUS indicator will be set to 1. In case BØGUS = 1 the routine 'INDFA' will be used to set error indicators.

INDFA: In case when bogus result is formed or when CPRA is performed, FA register no longer holds the flag bits of Operand A. FA is used to hold error indicators and return to TP. The corresponding bit in FA is set to 1 if the indicator is on.

Bit No. of FA	Indicator
1	GT (greater than)
2	EQ (equal)
3	LT (less than)
4	ØV (overflow)
5	FM (Flag Match)
6	UN (Underflow)
7	LSG (lost of significance)
8	ID (Invalid Data)

(7) TIMER: This routine is used to read out the internal clock of IBM 360/75 during execution. The timing of the simulation of one arithmetic order may then be known. The current time is printed in the following format: HH.MM.SS.TTT's where H stands for hour, M for minute, S for seconds and T for micro-second.

(ii) An overall flowchart of 'GATE' is provided

Symbols used in the flowchart are:

V: INBUS (50 bits)

M: M register

M_{i-j}: Bit i through j of M

UQ: UQ register

UH: UH register

Y: Y-input of SDS, this symbol is used for any one of the 4 stages of SDS.

P: borrow/carrier bits (64-bit string)

LM: LM register

UM: UM register

LS: LS register

US: US register

X: X-input to SDS, used for all 4 stages of SDS

S: S-input to SDS, " " " "

N: Negation control of SDS " " "

G: Interstage connection of SDS " "

T: T-output of SDS " " " "

Z: Z-output of SDS " " " "

P,B: Borrow/carry bits

II: Half-word integer holds EUL

FA: Flag register, usually holds the flag bits of operand A. In case of bogus result it holds error indicators.

GT: 'Greater than' indicator

EQ: 'Equal' indicator

LT: 'Less than' indicator

ØV: 'Overflow' indicator

UN: 'Underflow' indicator

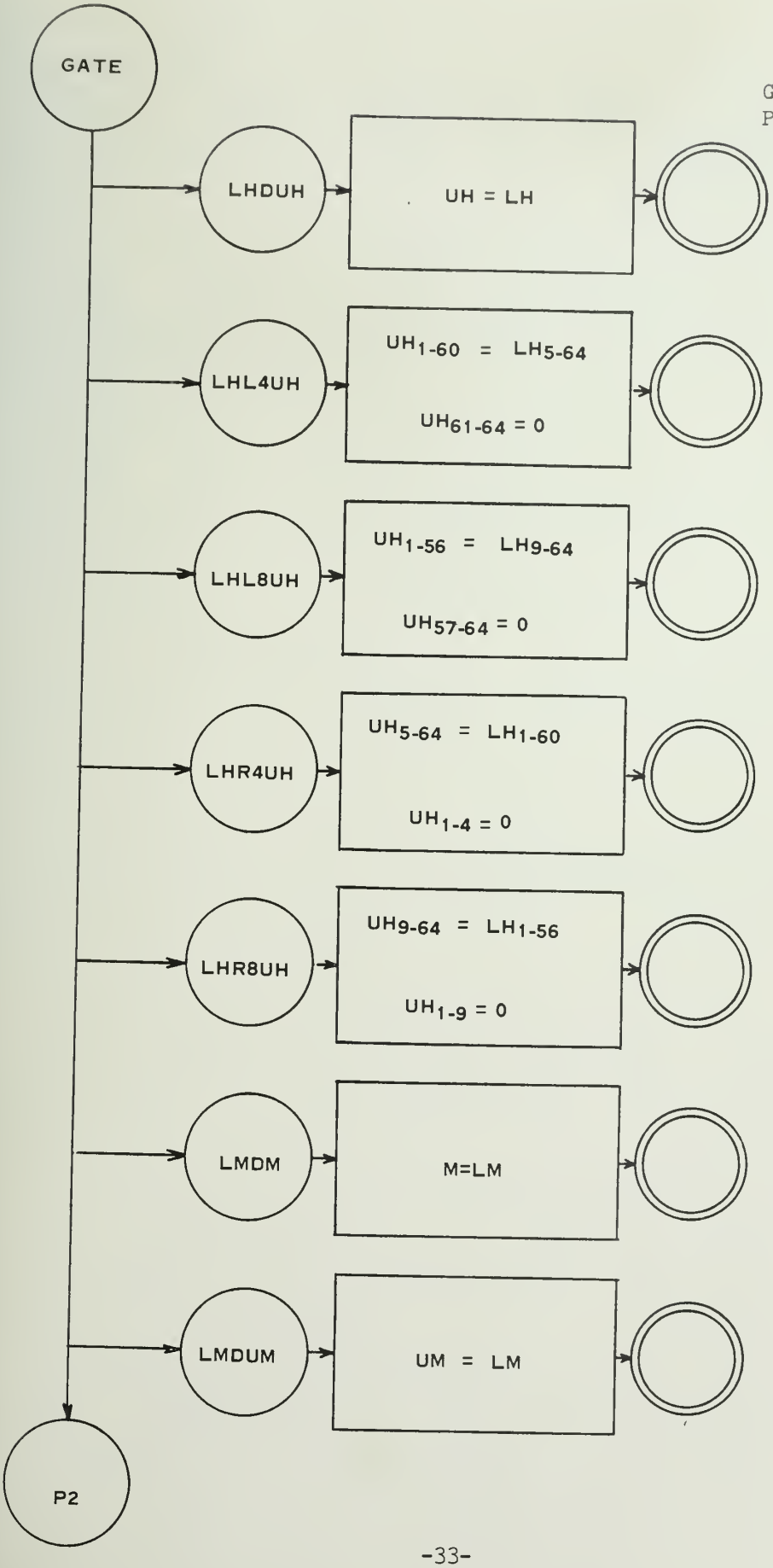
FM: 'Flag match' indicator

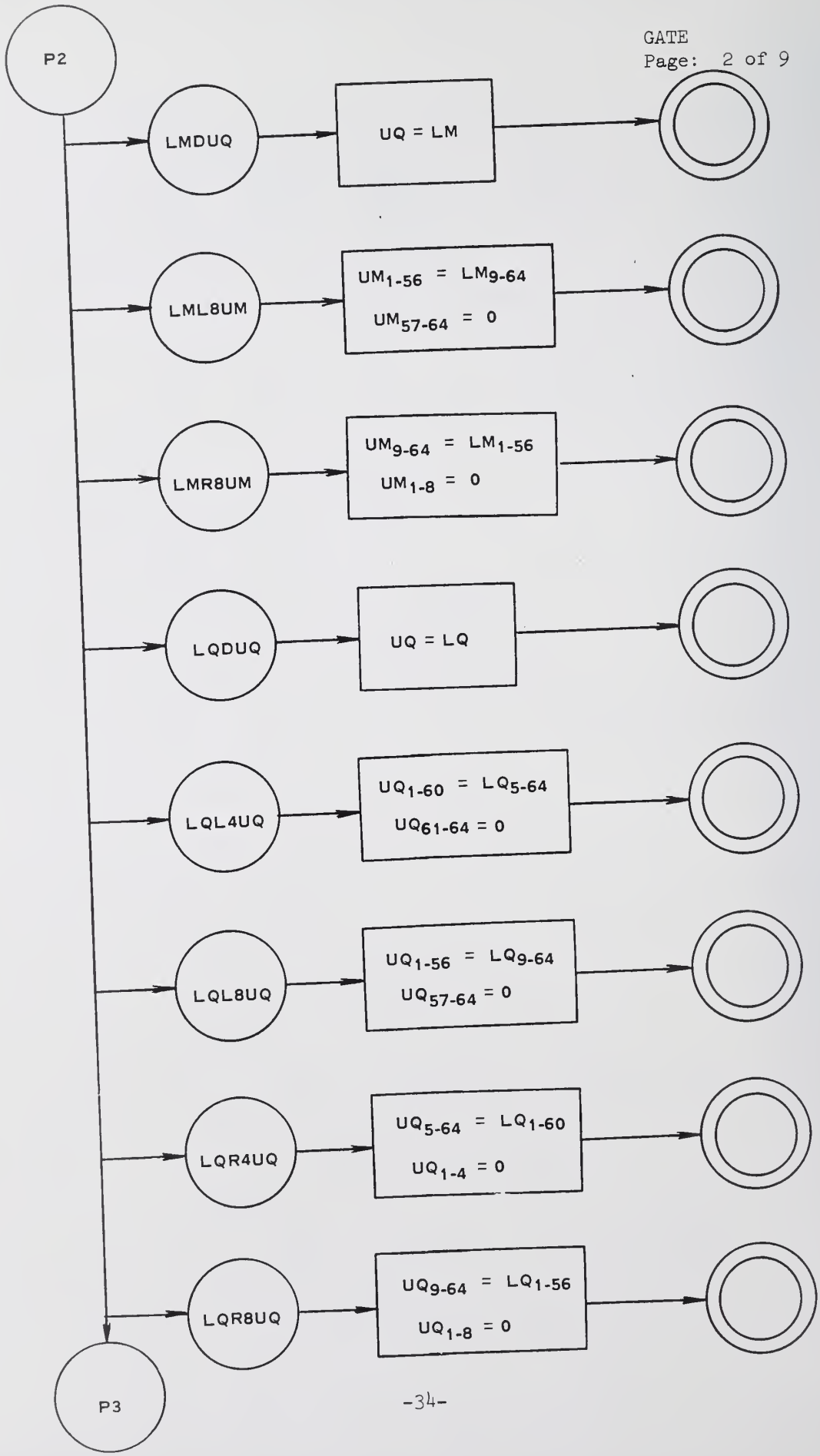
LSG: 'Loss of significance indicator

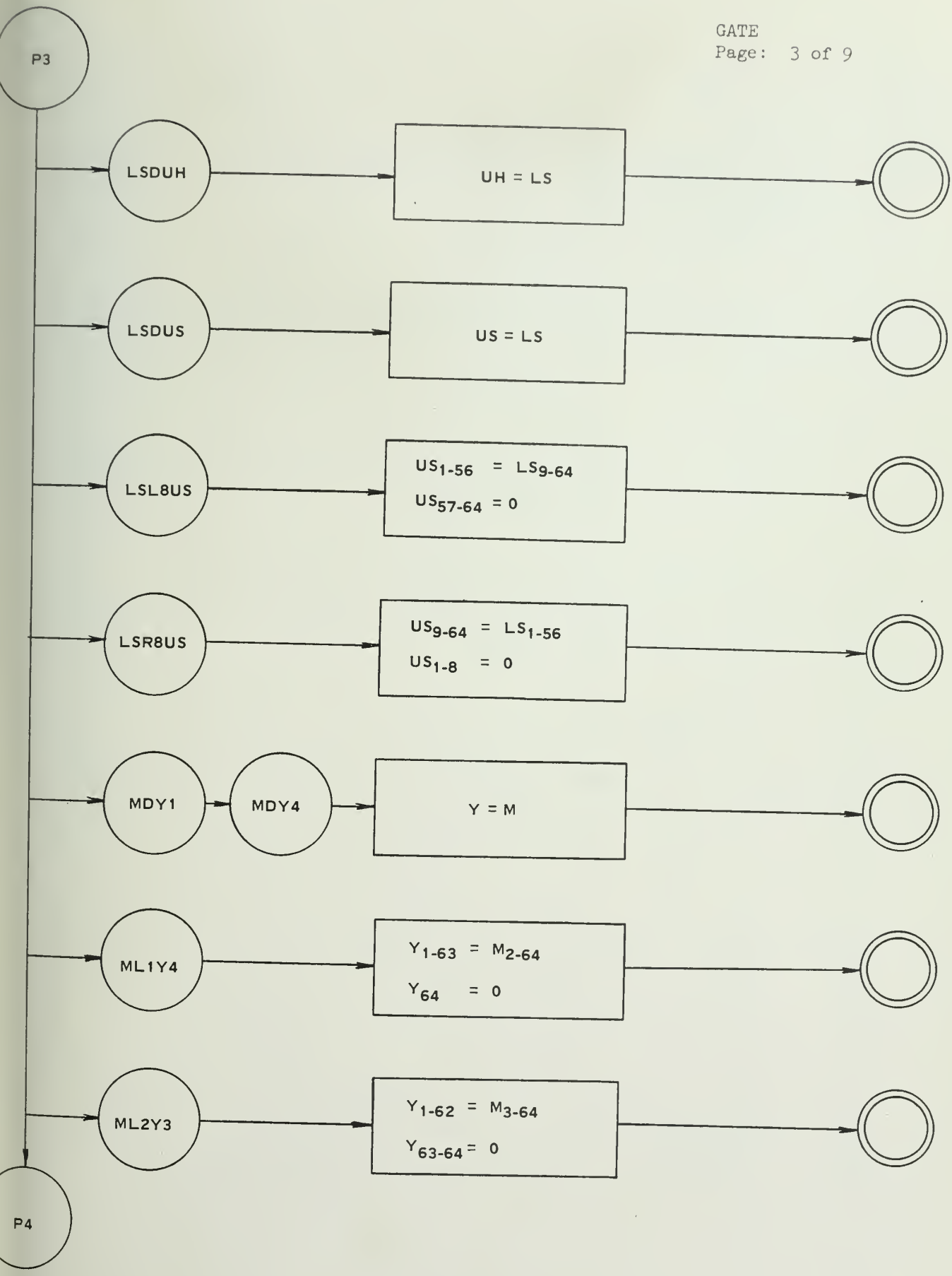
ID: 'Invalid data' indicator

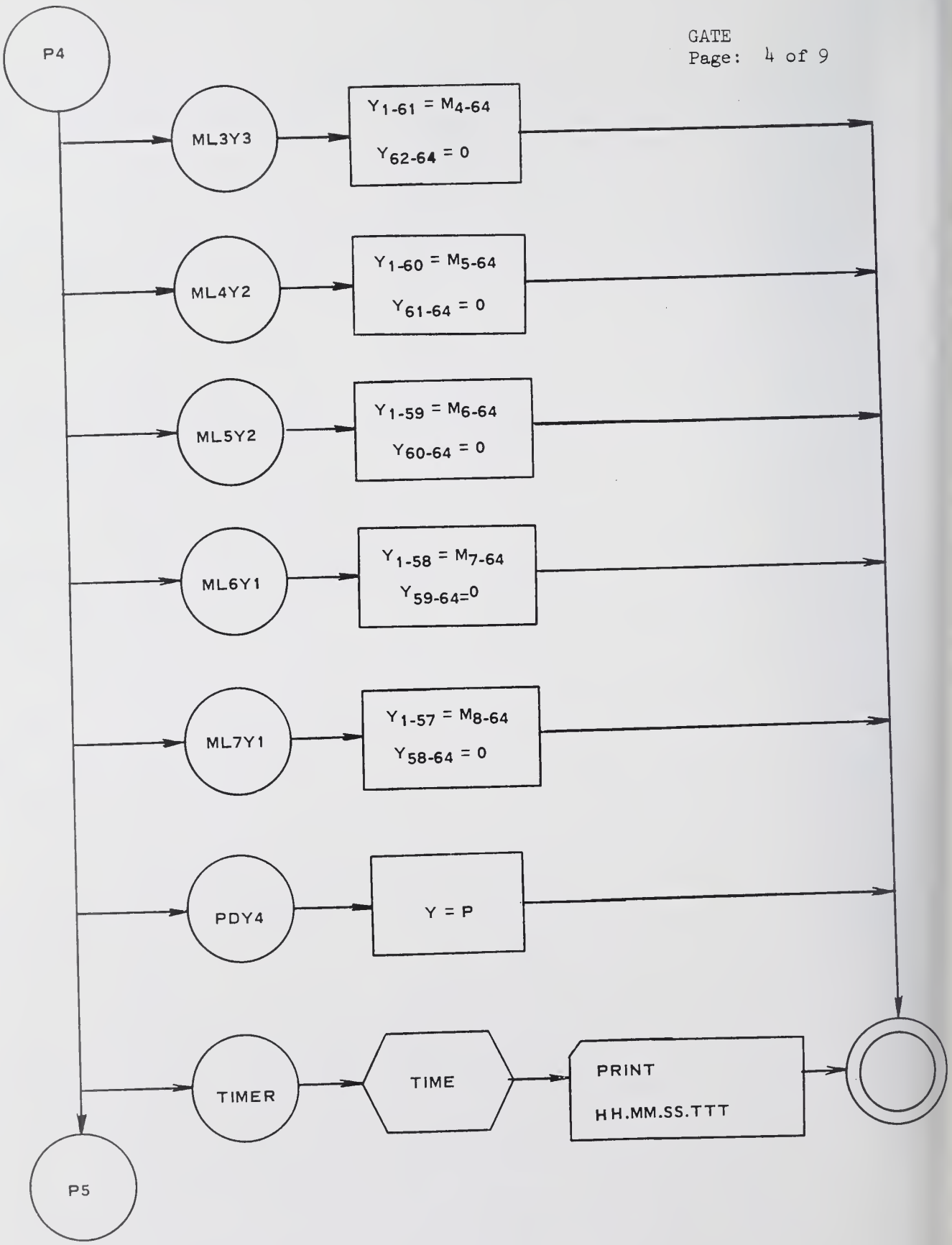
HH.MM.SS.TTT: Hour.Min.Sec. microsec

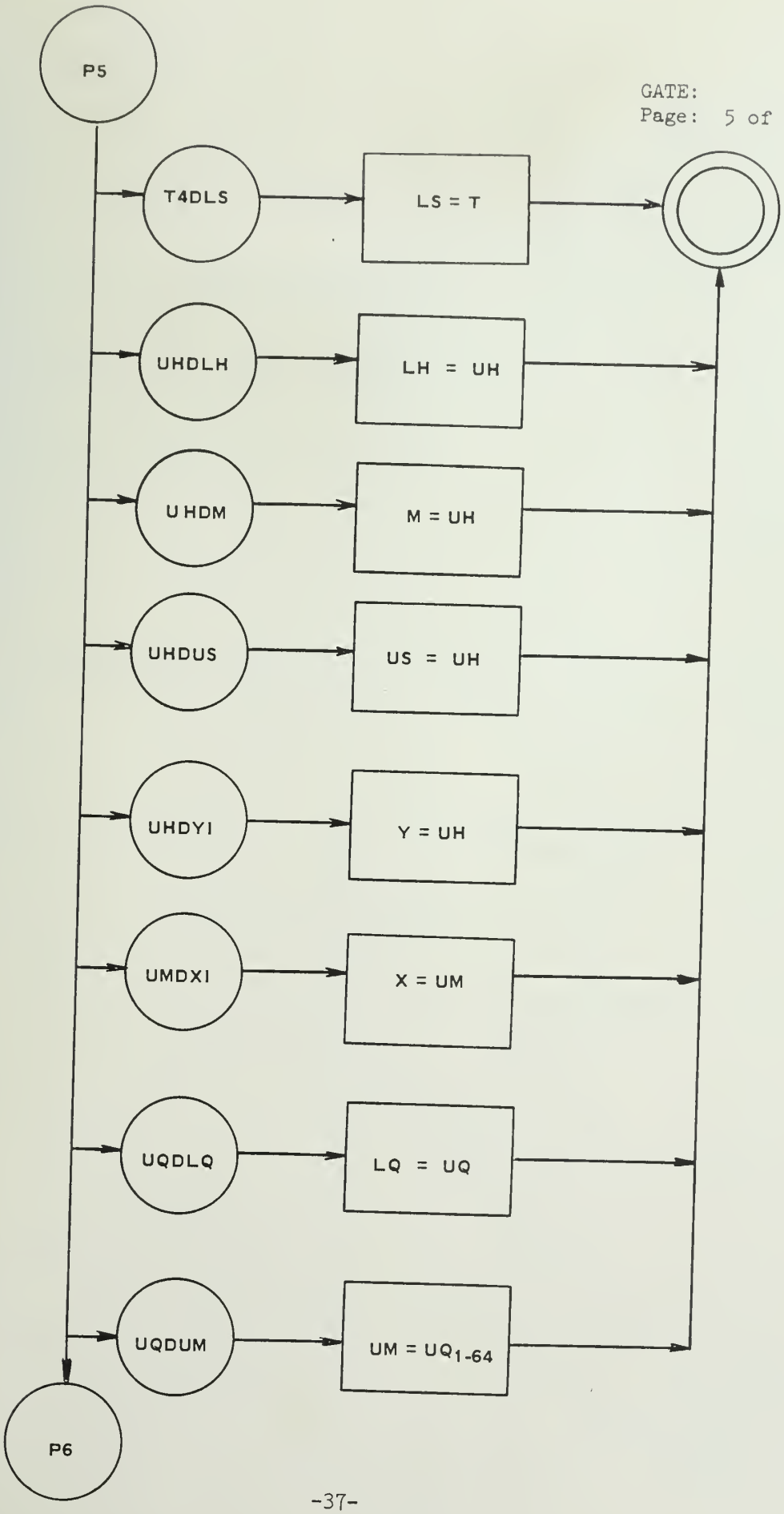
TIME: IBM 360/75 build-in function to read internal clock

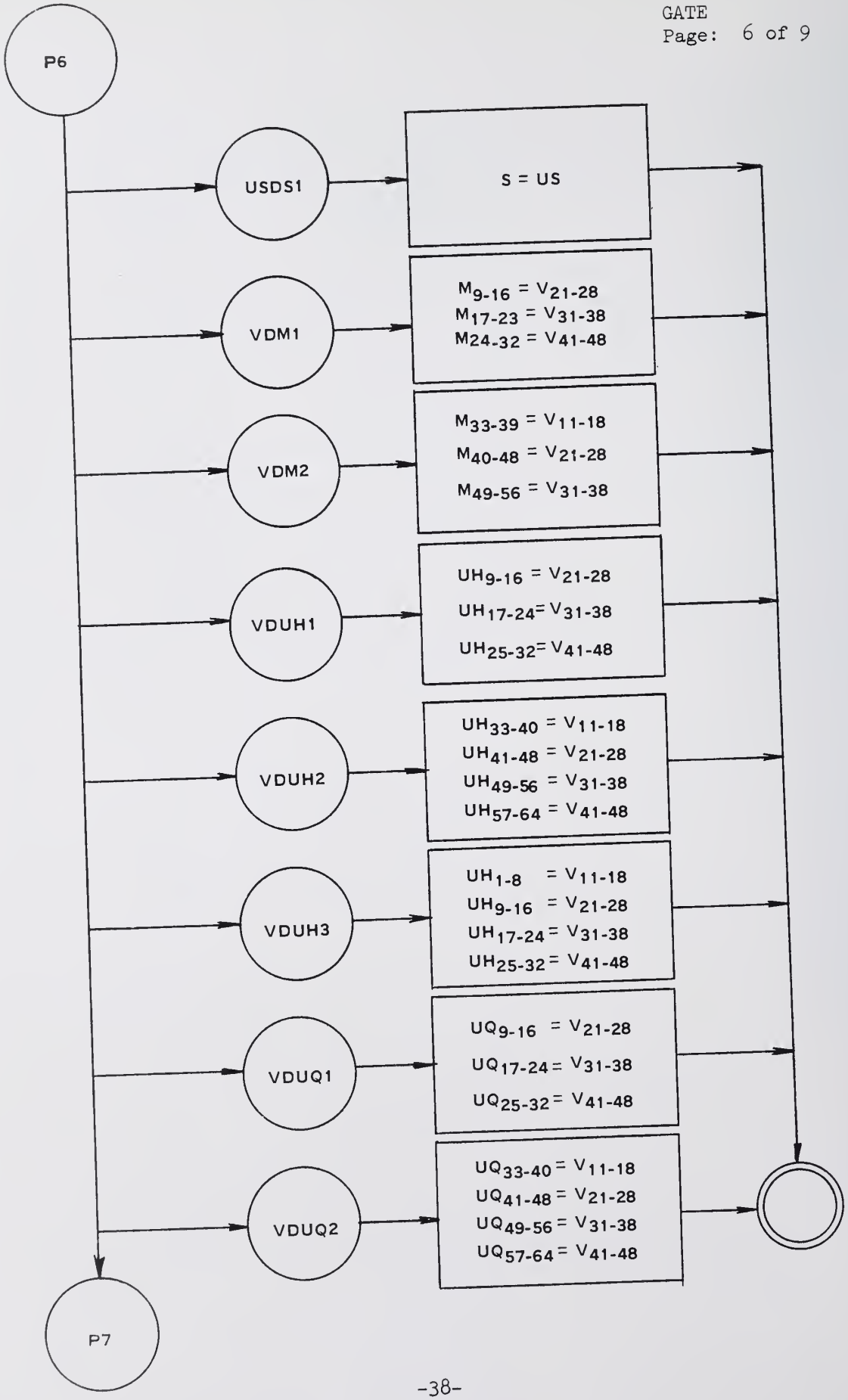

GATE
LHDUH
UH = LH
LHL4UH
UH1-60 = LH5-64
UH61-64 = 0
LHL8UH
UH1-56 = LH9-64
UH57-64 = 0
LHR4UH
UH5-64 = LH1-60
UH1-4 = 0
LHR8UH
UH9-64 = LH1-56
UH1-9 = 0
LMDM
M=LM
LMDUM
UM = LM
P2

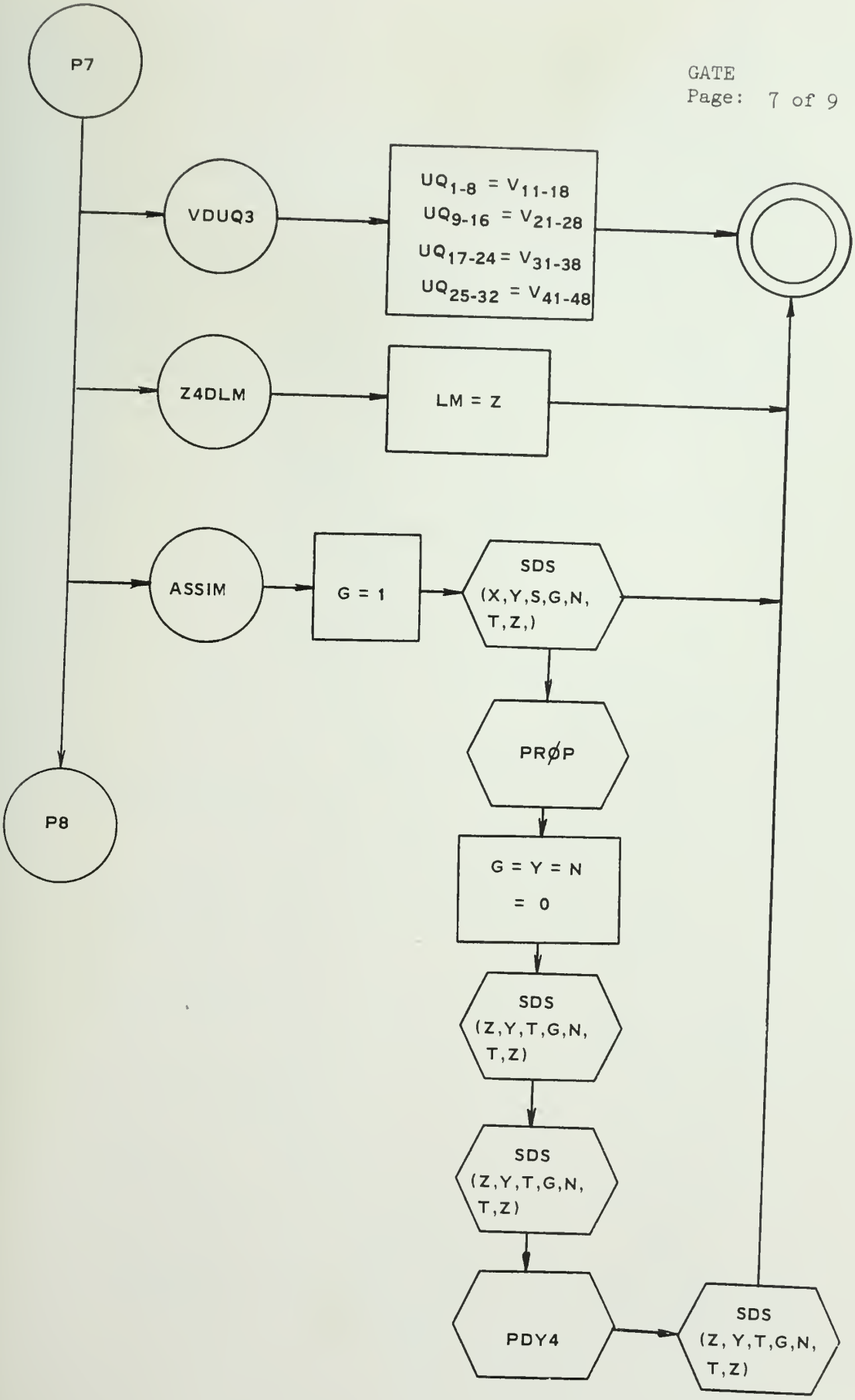

P2
LMDUQ
UQ = LM
LML8UM
UM1-56 = LM9-64
UM57-64 = 0
LMR8UM
UM9-64 = LM1-56
UM1-8 = 0
LQDUQ
UQ = LQ
LQL4UQ
UQ1-60 = LQ5-64
UQ61-64 = 0
LQL8UQ
UQ1-56 = LQ9-64
UQ57-64 = 0
LQR4UQ
UQ5-64 = LQ1-60
UQ1-4 = 0
LQR8UQ
UQ9-64 = LQ1-56
UQ1-8 = 0
P3

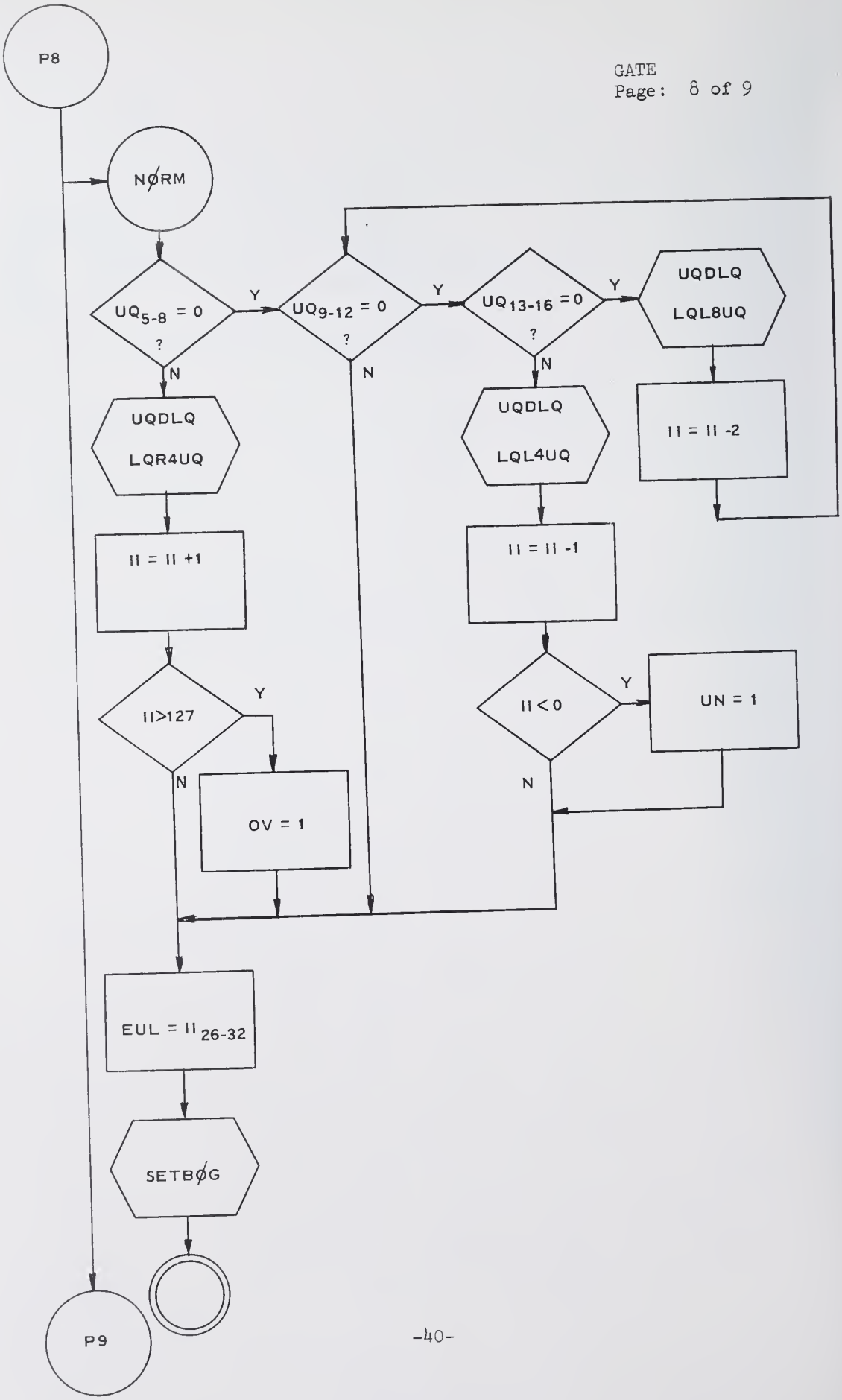

P3
LSDUH
UH = LS
LSDUS
US = LS
LSL8US
US1-56 = LS9-64
US57-64 = 0
LSR8US
US9-64 = LS1-56
US1-8 = 0
MDY1
MDY4
Y = M
ML1Y4
Y1-63 = M2-64
Y64 = 0
ML2Y3
Y1-62 = M3-64
Y63-64 = 0
P4

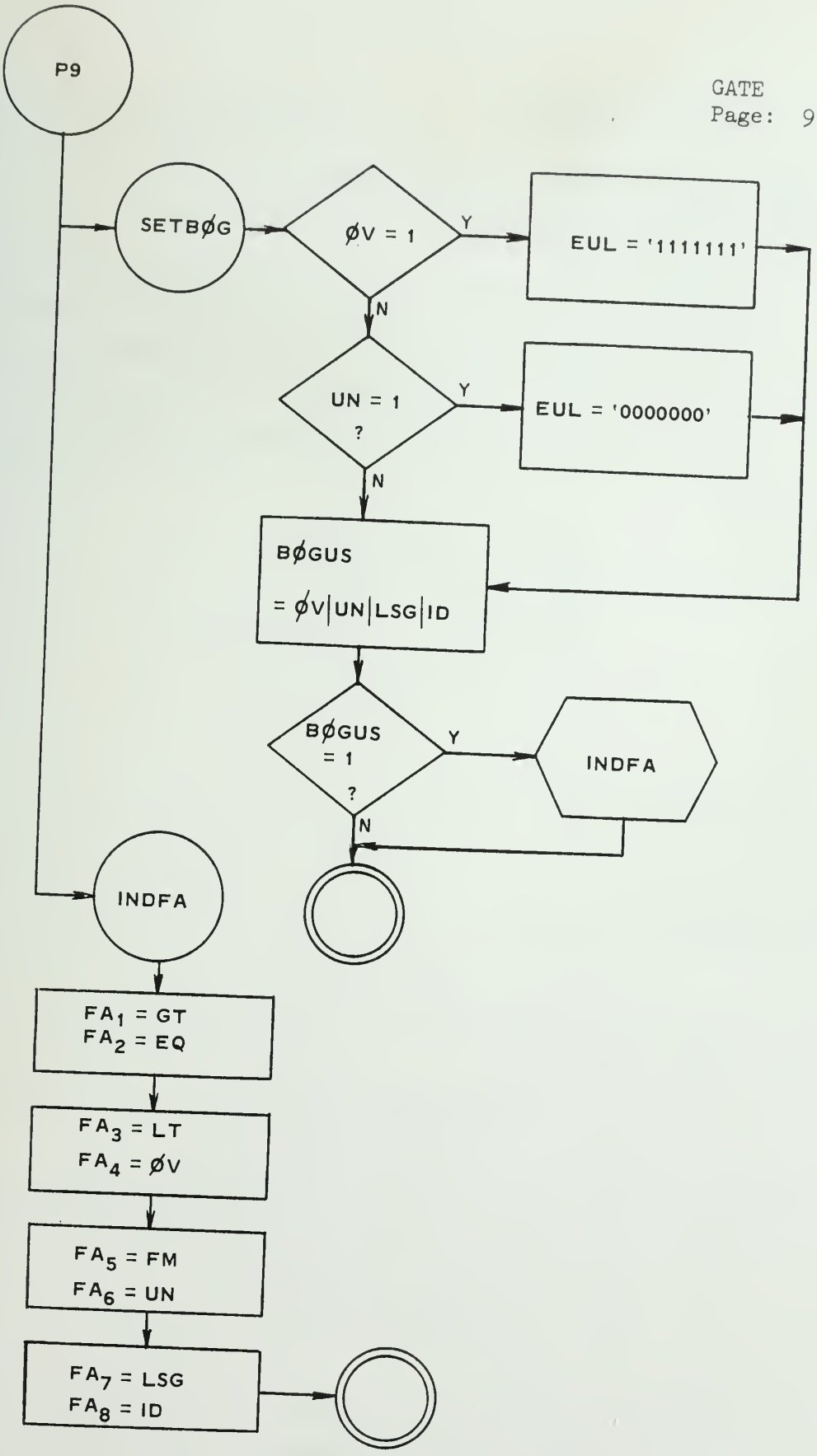

P4
ML3Y3
$Y_{1-61} = M_{4-64}$
$Y_{62-64} = 0$
ML4Y2
$Y_{1-60} = M_{5-64}$
$Y_{61-64} = 0$
ML5Y2
$Y_{1-59} = M_{6-64}$
$Y_{60-64} = 0$
ML6Y1
$Y_{1-58} = M_{7-64}$
$Y_{59-64} = 0$
ML7Y1
$Y_{1-57} = M_{8-64}$
$Y_{58-64} = 0$
PDY4
Y = P
TIMER
TIME
PRINT
HH.MM.SS.TTT
P5


P5
T4DLS
LS = T
UHDLH
LH = UH
UHDM
M = UH
UHDUS
US = UH
UHDYI
Y = UH
UMDXI
X = UM
UQDLQ
LQ = UQ
UQDUM
UM = UQ1-64
P6


P6
USDS1
S = US
VDM1
M9-16 = V21-28
M17-23 = V31-38
M24-32 = V41-48
VDM2
M33-39 = V11-18
M40-48 = V21-28
M49-56 = V31-38
VDUH1
UH9-16 = V21-28
UH17-24 = V31-38
UH25-32 = V41-48
VDUH2
UH33-40 = V11-18
UH41-48 = V21-28
UH49-56 = V31-38
UH57-64 = V41-48
VDUH3
UH1-8 = V11-18
UH9-16 = V21-28
UH17-24 = V31-38
UH25-32 = V41-48
VDUQ1
UQ9-16 = V21-28
UQ17-24 = V31-38
UQ25-32 = V41-48
VDUQ2
UQ33-40 = V11-18
UQ41-48 = V21-28
UQ49-56 = V31-38
UQ57-64 = V41-48
P7


P7
VDUQ3
UQ1-8 = V11-18
UQ9-16 = V21-28
UQ17-24 = V31-38
UQ25-32 = V41-48
Z4DLM
LM = Z
ASSIM
G = 1
SDS
(X,Y,S,G,N,
T,Z,)
PRØP
G = Y = N
= 0
SDS
(Z,Y,T,G,N,
T,Z)
SDS
(Z,Y,T,G,N,
T,Z)
PDY4
SDS
(Z,Y,T,G,N,
T,Z)
P8


P8
NØRM
UQ5-8 = 0 ?
Y
N
UQ9-12 = 0 ?
Y
N
UQ13-16 = 0 ?
Y
N
UQDLQ LQL8UQ
UQDLQ LQR4UQ
UQDLQ LQL4UQ
II = II -2
II = II +1
II = II -1
II>127
Y
N
II<0
Y
N
UN = 1
OV = 1
EUL = II26-32
SETBØG
P9


P9
SETBØG
ØV = 1
Y
EUL = '1111111'
N
UN = 1
?
Y
EUL = '0000000'
N
BØGUS
= ØV|UN|LSG|ID
BØGUS
= 1
?
Y
INDFA
N
INDFA
FA1 = GT
FA2 = EQ
FA3 = LT
FA4 = ØV
FA5 = FM
FA6 = UN
FA7 = LSG
FA8 = ID

IV. The Signed-Digit Subtractor

1. Name: SDS

2. Functional Description:

The adder used in the Illiac III is actually a signed-digit subtractor. It includes the facility for postponing borrow propagation. It will perform both addition and subtraction. The subtrahend is in conventional binary form, the minuend is in SDS format: each digit of the minuend is of the form $S_i - X_i$, where X_i is interpreted as a magnitude, 1 or 0 and S_i is a sign, 0 = +, 1 =-1. The output of the subtractor is in this same SDS format.

The SDS format digits are represented as follows:

S_i	X_i	Digital Value
0	0	+0
0	1	+1
1	0	-0
1	1	-1

3. Formal Calling Sequence:

CALL SDS (X, S, Y, G, NEG, T, Z)

4. Formal Parameter Description:

4.1 Input Parameters:

S: Sign of minuend digits

X: Magnitude of minuend digits

Y: Subtrahend in conventional binary form

G: Gate on interstage connection (not a propagation borrow/carry)

NEG: Control to complement T

NEG = 0 → T not complemented

NEG = 1 → T complemented

4.2 Output Parameters:

T: Sign digits of difference

Z: Magnitude digits of difference

All input and output parameters are 64 digits.

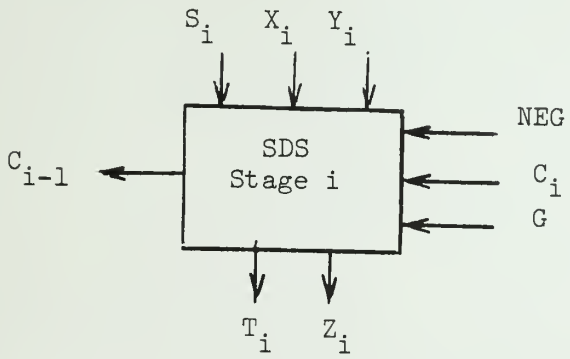
5. Implicit Parameter Description:

6. Subroutines Used: The external variable PRINT is used to control the printed output of this routine. If PRINT = 0, nothing will be printed from this routine.

7. Operational Description

7.1 English Text:

Each stage of the signed-digit subtractor has 3-input and 2-output together with an interstage connection and 'NEG' control line.



This routine is used to evaluate the following equations:

$C_i = (S_{i+1} X_{i+1} \vee \overline{X}_{i+1} Y_{i+1}) \cdot G$ $\qquad i = 1, 2 \ldots 64$ (bits)

$T_i = C_i + NEG$

$Z_i = C_i + (X_i + Y_i)$

$+ \rightarrow$ module 2- sum. (Exclusive -or)

In case when $S_1 = X_1 =$ '1'B then $T_1 = \overline{T}_1$

Since all input and output parameters are defined as 64-bit string it is a straightforward bit manipulation.

Whenever this routine is called, the parameter PRINT will be checked. If PRINT=1, the bit-strings of X, S, Y, G, N, T, Z will be printed out. Otherwise, nothing will be printed.

7.2 Flowchart

Flowchart is attached

Symbols used in the flowchart are:

S: Sign of minuend in SDS format

C_{i-j}: ith to jth bits of C.

X: Magnitude of minuend in SDS format

Y: Subtrahend in conventional format

T: Sign of result in SDS format

Z: Magnitude of result in SDS format

N: Negation control

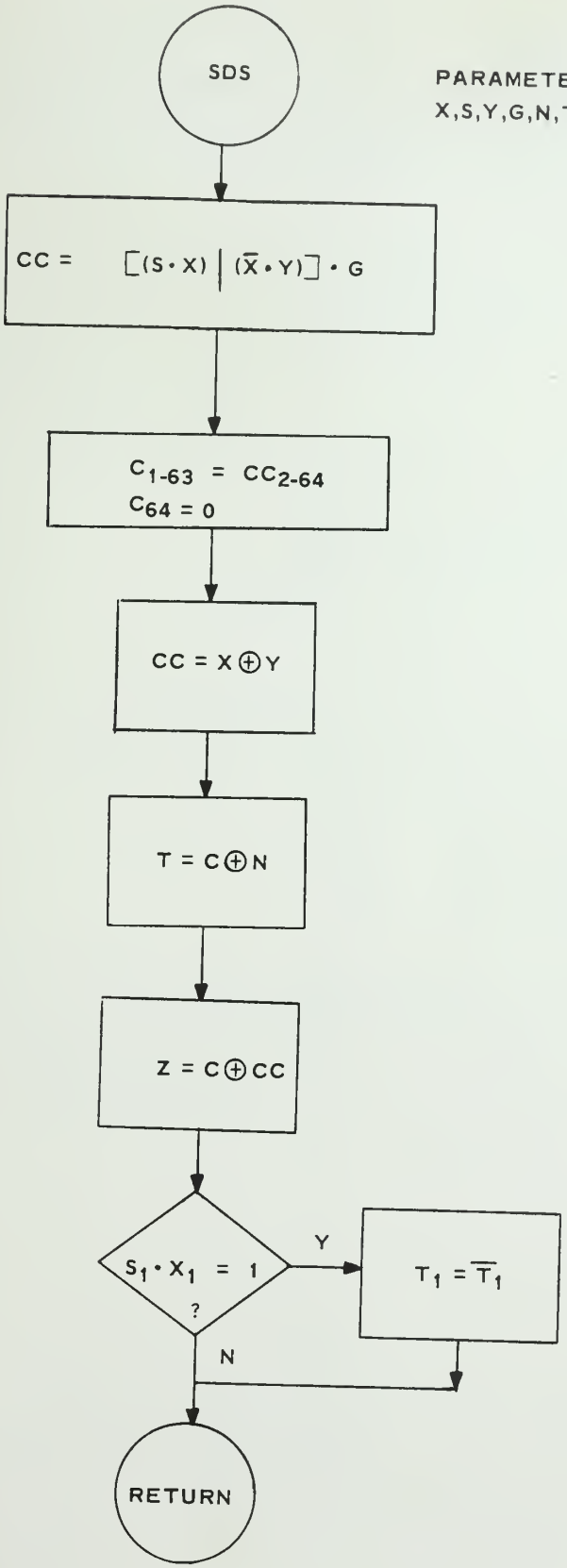
G: Gate on interstage connections

C: Interstage connections

CC: Temporary storage

All variables are 64-bit string and used for any one of the 4 stages of SDS).

8. Error Conditions: None


SDS
PARAMETERS:
X,S,Y,G,N,T,Z
CC = [(S • X) | (X̄ • Y)] • G
C1-63 = CC2-64
C64 = 0
CC = X ⊕ Y
T = C ⊕ N
Z = C ⊕ CC
S1 • X1 = 1
?
Y
T1 = T̄1
N
RETURN

V. Propagation Logic

1. Name: PRØP

2. Functional Description:

This routine is used to generate the borrow/carry digits, B, which will be used to assimilate the result of SDS from SD format to conventional format. B bits are defined as:

$$B_{i-1} = B_i \cdot \overline{Z}_i \vee T_i \cdot Z_i \quad (i = m, m_{i-1}, \ldots -0)$$

where Z_i = ith magnitude bit

T_i = ith sign bit

i = Number of the least significant digit, in this simulation. It is 64.

3. Formal Calling Sequence: CALL PRØP

4. Formal Parameter Description: None

5. Implicit Parameter Description:

All B, T, Z are defined as external variables, so they don't have to be passed as parameter, but B is changed after execution of this routine.

6. Subroutines Used: None

7. Operational Description:

7.1 English Text:

This routine is used to generate the borrow bits, since B_i depends upon B_{i+1}, the equation mentioned above cannot be evaluated by simple bit-string manipulation. The build-in function SUBSTR of PL/1 is used. It will evaluate every bit of B by the equation from the least significant bit to most significant bit.

7.2 Flowchart

Flowchart is attached:

Symbols used in the flowchart are:

B: Borrow bits (64-bit string)

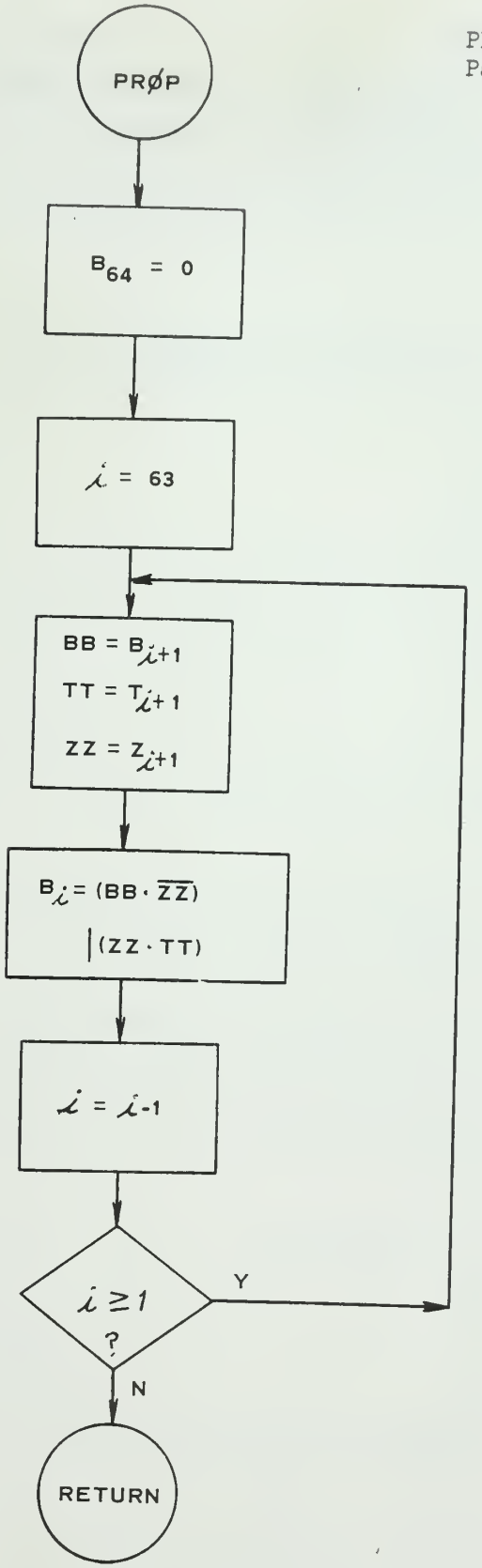
T: Sign bits of result from SDS (64-bit string)

Z: Magnitude bits of result from SDS (64 bit string)

BB , TT, ZZ: Temporary storage (1 bit)

B_i: ith bit of B

8. Error Condition: None


PRØP
$B_{64} = 0$
$i = 63$
BB = B_{i+1}
TT = T_{i+1}
ZZ = Z_{i+1}
B_i = (BB · $\overline{ZZ}$)
| (ZZ · TT)
$i = i-1$
$i \geq 1$
?
Y
N
RETURN

VI. Arithmetic Orders

Every arithmetic order is simulated separately. ADD, SUB and CPRA are quite similar and will be described in one section. There are three types of data conversion: CVL, CVF and CVD which will be explained in one section. MPY, DIV and PØLY will be explained as individual procedures.

6.1 Addition, Subtraction and Comparison

1. Name: ADDX
   or SUB
   or CPRA

2. Functional Description:

This routine is used to simulate floating point addition, subtraction or comparison.

If the exponent of sum or difference is greater than 63, overflow occurs. The bogus result will be the maximum floating point number that can be represented, with correct sign.

If the exponent of sum or difference is less than -64, underflow occurs. The bogus result will be the minimum floating point number that can be represented, with correct sign.

If the fraction of sum or difference is zero but the exponent is not -64, then loss of significance occurs. The bogus will be zero.

For comparison, the EQ, GT, LT or FM will be set according to the result of comparison.

3. Formal Calling Sequence:

   CALL ADDX
   or CALL SUB
   or CALL CPRA

4. Formal Parameter Description: None

5. Implicit Parameter Description:

The augend (minuend) is taken from UQ register, the addend (subtrahend) is taken from UH register. The sum or difference is held in UQ register. For CPRA or bogus results, FA holds the comparison or bogus result indicators, otherwise FA is unchanged.

6. Subroutines used:

ASSIM, INDFA, LHR4UH, LHR8UH, LMDUQ, LQR4UQ, LQR8UQ, MDY1, MDY4, NØRM, SDS, UHDLH, UHDM, UMDX1, UQDLQ, UQDUM, USDS1, Z4DLM.

7. Operational Description:

7.1 English Text:

(1) The first operand (augend or minuend) is taken from the UQ register (fraction). Its exponent is in EUU, the sign bit in SIGNA, and the flag bits in FA. For the second operand (addend or subtrahend), the fraction is in the UH register, the exponent in EUM, the sign bit in SIGNA, and the flag bits in FB.

(2) The indicator PLUS, MINUS or CPA is set according to whether the instruction is ADD, SUB or CPRA.

(3) DEXP is the difference of the two exponents. When DEXP is either greater than 0 or less than 0, the sign of the result is predictable. When DEXP is equal to zero, in some cases, the sign of the result is also predictable. Whenever the sign of result is predictable, the indicator SDB will set to 1. For CPRA, if SDB = 1, GT or LT can be assigned without actually doing the subtraction. The following is a table for the prediction of SR.

DEXP	SIGNA ≡ SIGN B	Arithmetic Order	SDB	SR
> 0	Yes or No	Any	1	SIGNA
< 0	Yes or No	Any	1	SIGNB
= 0	Yes	ADD	1	SIGNA
		SUB	0	0
		CPRA	0	0
	No	ADD	0	0
		SUB	1	SIGNA
		CPRA	1	SIGNA

(4) After SDB and SR have been set, if DEXP > 13 then the second operand is too small compared to the first operand. The result (sum or difference) will be the first operand, i.e. the result is the contents of UQ itself.

At point (1) for $0<DEXP\leq 13$, the second operand will be right shifted and DEXP will decrease accordingly until DEXP = 0. (Aligned to the same binary point).

At point (2) for $-13\leq DEXP<0$, the second operand will shift left, the DEXP will increase accordingly until DEXP=0.

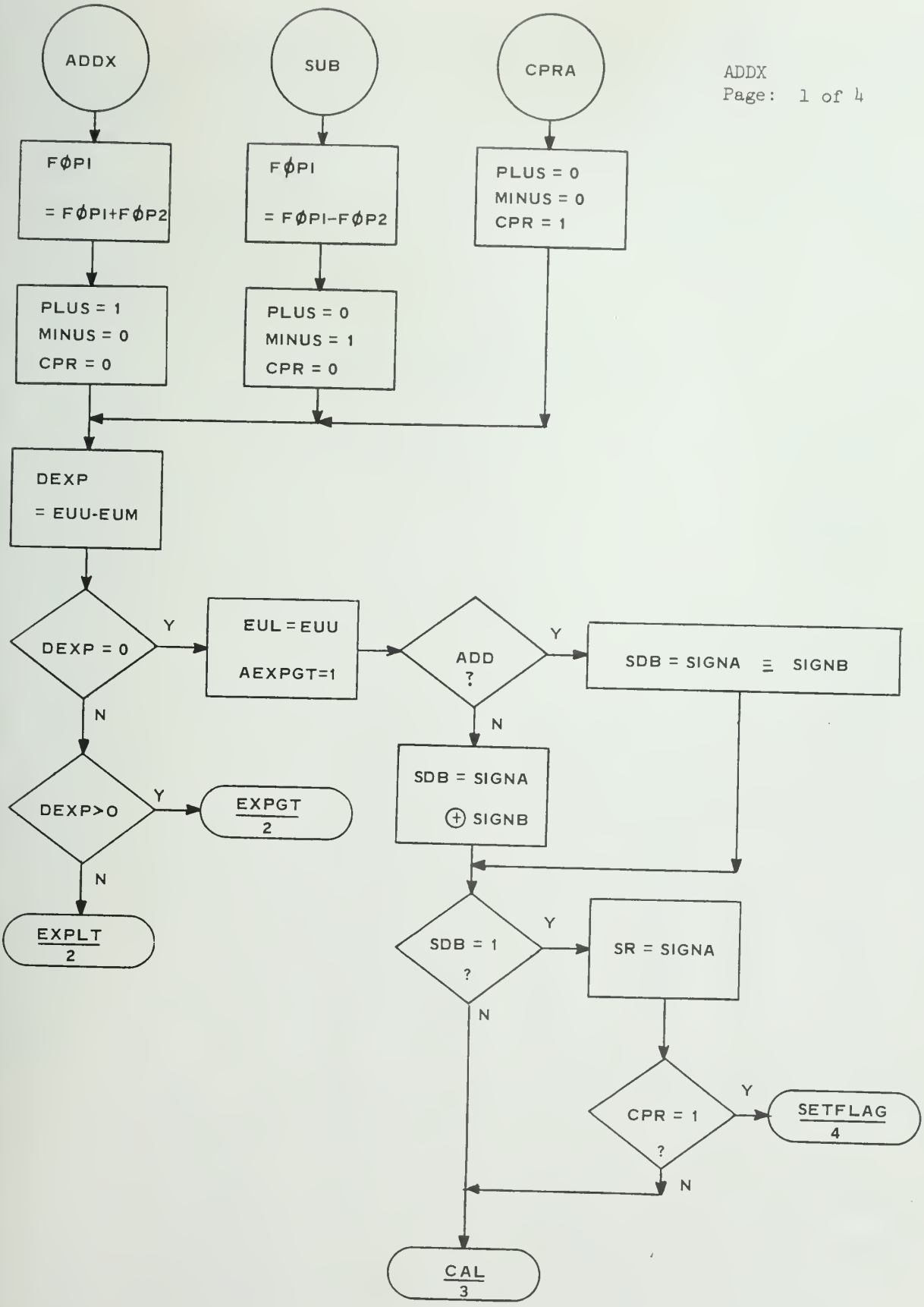
At point (3) for DEXP<-13, the result will be the second operand, so the contents of UH will be transmitted into UQ register by passing through SDS stage 4.

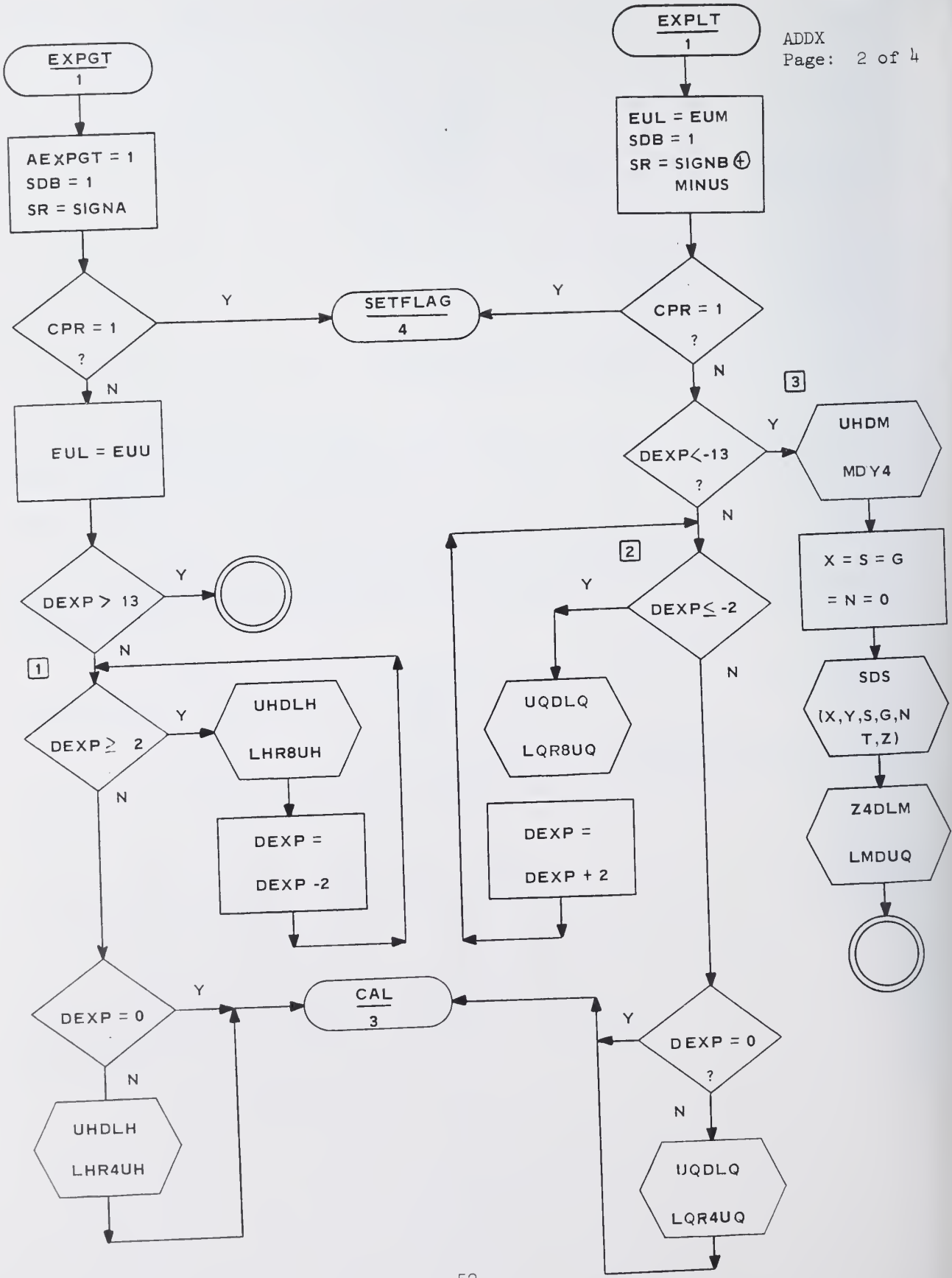
(5) At point (4), the negation control NEGØ and N (SDS stage 1) is set according to the instruction invariant. The equations implemented are:

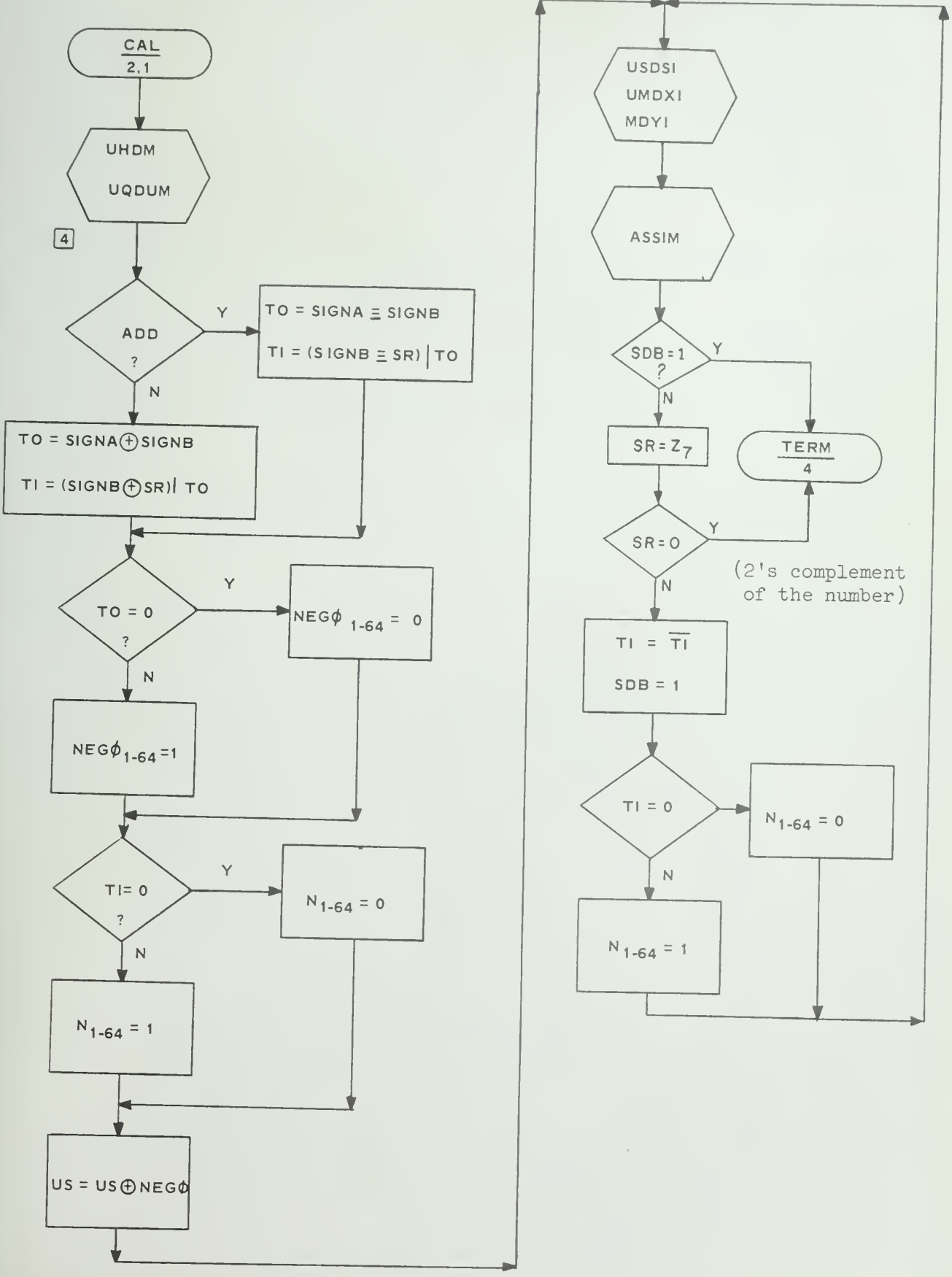
NEGØ = (SIGNA ≡SIGNB) . ADD |(SIGNA ⊕ SIGNB). (SUB|CPRA)

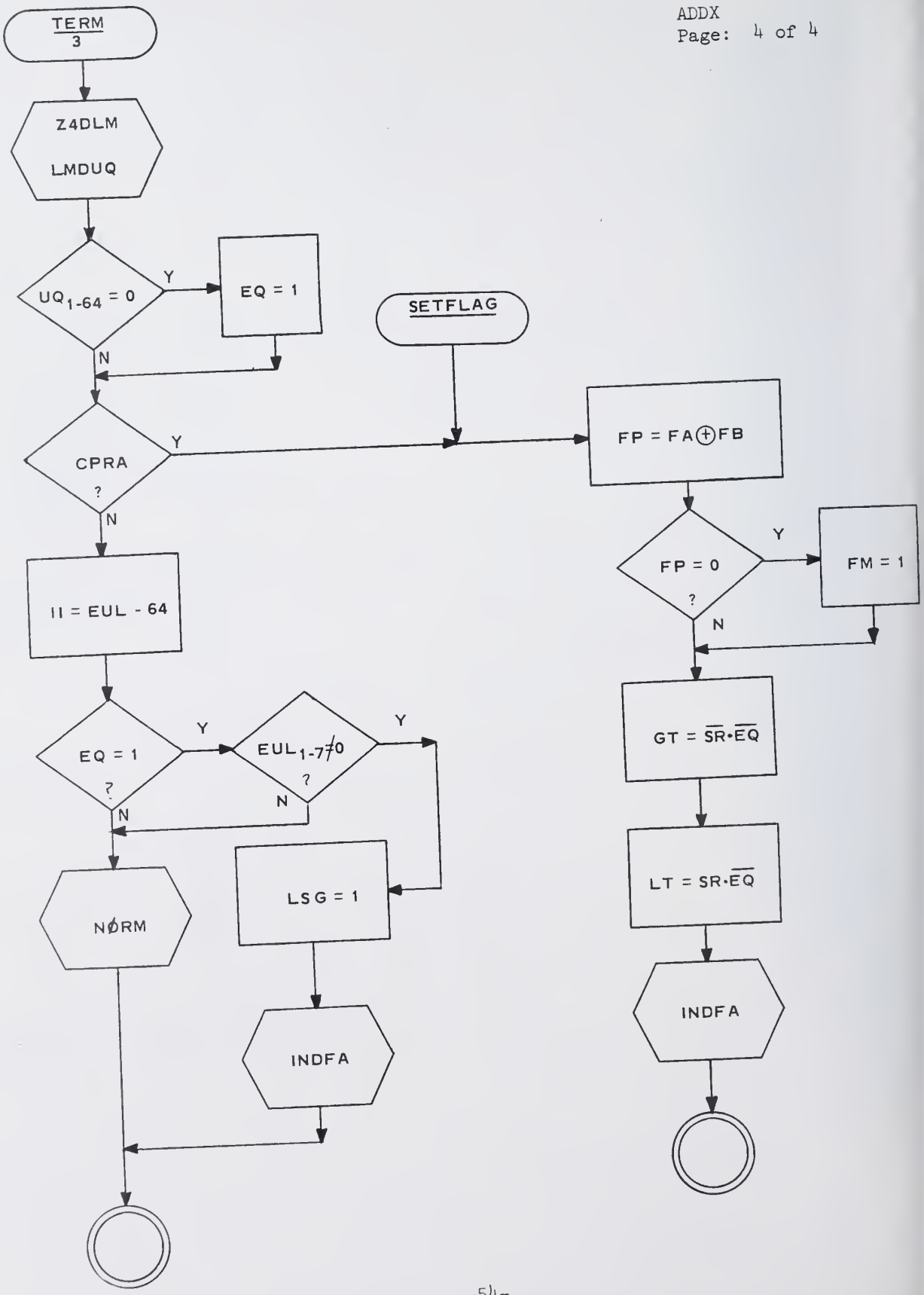
N = [(SIGNA ≡ SIGNB)| (SIGNB ≡SR)] ADD

|[(SIGNA ⊕ SIGNB) |(SIGNB ⊕ SR)] (SUB|CPRA)


ADDX
SUB
CPRA
FØPI
= FØPI+FØP2
FØPI
= FØPI-FØP2
PLUS = 0
MINUS = 0
CPR = 1
PLUS = 1
MINUS = 0
CPR = 0
PLUS = 0
MINUS = 1
CPR = 0
DEXP
= EUU-EUM
DEXP = 0
Y
EUL = EUU
AEXPGT=1
ADD
?
Y
SDB = SIGNA ≡ SIGNB
N
N
SDB = SIGNA
⊕ SIGNB
DEXP>0
Y
EXPGT
2
N
EXPLT
2
SDB = 1
?
Y
SR = SIGNA
N
CPR = 1
?
Y
SETFLAG
4
N
CAL
3


EXPGT
1
AEXPGT = 1
SDB = 1
SR = SIGNA
CPR = 1
?
Y
N
SETFLAG
4
EUL = EUU
DEXP > 13
Y
N
1
DEXP ≥ 2
Y
N
UHDLH
LHR8UH
DEXP =
DEXP -2
DEXP = 0
Y
N
UHDLH
LHR4UH
CAL
3
EXPLT
1
EUL = EUM
SDB = 1
SR = SIGNB ⊕
MINUS
CPR = 1
?
Y
N
3
DEXP<-13
?
Y
N
UHDM
MDY4
X = S = G
= N = 0
SDS
(X,Y,S,G,N
T,Z)
Z4DLM
LMDUQ
2
DEXP≤ -2
Y
N
UQDLQ
LQR8UQ
DEXP =
DEXP + 2
DEXP = 0
?
Y
N
UQDLQ
LQR4UQ


CAL
2,1
UHDM
UQDUM
4
ADD
?
Y
TO = SIGNA ≡ SIGNB
TI = (SIGNB ≡ SR) | TO
N
TO = SIGNA ⊕ SIGNB
TI = (SIGNB ⊕ SR) | TO
TO = 0
?
Y
NEGϕ 1-64 = 0
N
NEGϕ 1-64 = 1
TI = 0
?
Y
N 1-64 = 0
N
N 1-64 = 1
US = US ⊕ NEGϕ
USDSI
UMDXI
MDYI
ASSIM
SDB = 1
?
Y
N
SR = Z7
TERM
4
SR = 0
Y
N
(2's complement
of the number)
TI = $\overline{TI}$
SDB = 1
TI = 0
N 1-64 = 0
N
N 1-64 = 1


TERM 3
Z4DLM
LMDUQ
UQ1-64 = 0
Y
EQ = 1
N
SETFLAG
CPRA ?
Y
N
FP = FA⊕FB
FP = 0 ?
Y
FM = 1
N
I1 = EUL - 64
GT = SR̄·EQ̄
EQ = 1 ?
Y
EUL1-7≠0 ?
Y
N
N
LT = SR·EQ̄
NORM
LSG = 1
INDFA
INDFA

Since NEGØ and N are defined to be 64-bit strings, but both SIGNA and SIGNB are 1-bit, temporary storage TO and T1 are introduced to set NEGØ and N.

(6) The result US ⊕ NEGØ is used as the S-input to SDS. The contents of UQ register is used as X-input (UQDUM, UMDX1) and the contents of UH register is used as Y-input to SDS, Stage 1. N is used to control add or subtract and then the 'ASSIM' routine is used to perform the actual addition or subtraction.

(7) In case when SDB = 0 and SR = 1, that means the N set above is wrong. So reset N just by its complement and repeat step (6).

(8) The assimilated result is in the UQ register. If the content of UQ is 0, the EQ indicator is set to 1. In case UQ = 0 but EUL ≠ 0 then loss of significance occurs, LSG is set to 1, and the routine 'INDFA' is used to load FA.

In other cases, 'NØRM' is used to normalize the result. If overflow or underflow occurs during normalization, it will be taken care by 'NØRM'.

(9) For CPRA, FA and FB is compared to set FM.

The comparison is done by exclusive OR (A temporary storage, FP, of 8-bits is needed to hold the result of the exclusive -OR due to the restriction of the implementation of bit manipulation in PL/1). GT and LT are set by SR. Routine 'INDFA' is used to load FA.

7.2 Flowchart

Flowchart is attached.

Symbols used in the flowchart are:

PLUS: = 1 for ADD (1 bit)

MINUS: = 1 for SUB (1 bit)

CPR: = 1 for CPRA (1 bit)

DEXP: Difference between EUU and EUM

SDB: Sign-Digit-Bypass, = 1 if SR is predictable.

II: EUL extended to 16 bit.

TO: Used to set NEGØ. (1 bit)

T1: Used to set N (in stage 1) (1 bit)

$\overline{T1}$: Not T1

FP: Temporary storage to hold FA ⊕ FB (8 bit).

GT: 'greater than' indicator

LT: 'Less than' "

EQ: 'Equal' "

LSG: 'loss of significance' indicator.

8. Error Condition:

In case of overflow, the bogus result is the maximum floating number with correct sign. For underflow, the bogus result is the minimum floating number with correct sign. For lost of significance, the bogus result is 0.

6.2 Multiplication:

1. Name: MPY

2. Functional Description:

This routine is used to simulate the fixed point and floating point multiplication.

For fixed point multiplication, if the product is greater than $2^{31}-1$ or less than -2^{31}, an overflow will occur. The result returned to TP will have the most significant bits in UQ_{1-32}, the least significant bit in UQ_{33-64}.

For floating point numbers, if the exponent of the product is greater than 63, an overflow will occur. The bogus result will be the maximum floating point number which can be represented, with the correct sign bit. If the exponent of the product is less than -64, an underflow will occur, the bogus result will be the minimum number which can be represented, with correct sign bit.

3. Formal Calling Sequence:

CALL MPY

4. Formal Parameter Description: None

5. Implicit Parameter Description:

The multiplier and multiplicand are taken from UQ and UH register (exponent in EUU and EUM), respectively. Signs are taken from SIGNA and SIGNB

The exponent of the product is in EUL, the fraction in UQ and sign in SR.

6. Subroutines Used:

UHDLH, LHR8 UH, UHDM, UQDLQ, UMDX1, SDS, USDS1, ML7YL, ML6Y1, ML5Y2, ML4Y2, ML3Y3, ML2Y3, ML1Y4, MDY4, T4DLS, Z4DLM, LQR8UQ, LSR8US, LMR8UM, MEXTPRE, LSDUS, LMDUM, ASSIM, NØRM, BØGUS, LMDUM.

7. Operational Description:

7.1 English Text

(1) Sign of the product will be the exclusive ØR of the sign of the multiplicand and multiplier (SIGNA ⊕ SIGNB).

(2) For floating point number, as at point (1) the fraction of the multiplier is in the UQ register, the exponent in EUU. The fraction of multiplicand is taken from UH register with the exponent in EUM register.

If either the multiplicand or multiplier is zero, the product will be zero.

The exponent of the product is the sum of the exponent of multiplicand and the multiplier. (II is the sum of exponent extended from 7-bits to 16-bits. Since the first bit of EUU and EUM are sign bits, 128 must be subtracted): If II> 63 overflow occurs. If II<-64 underflow occurs. The multiplicand is transmitted from to UH register to the M register. The total multiply cycles needed is 7.

(3) For fixed point multiplication, the multiplier is taken from UQ register, bits 33 through 64; the multiplicand is taken from UH register bits 1 through 32.

In order to share the same logic as for floating point multiplication, the UH register is right shifted 8 bits because for floating point case, UH contains all zeroes in the first byte for positive multiplicand. For negative multiplicand, bit 1-8 of UH must be set to 1, as shown at point (3).

The multiplicand is then loaded into M register from UH register. For short fixed point multiplication, the multiply cycle needed is 2; for long fixed point, it is 4.

(4) From here on, the following steps are common for both floating point number and fixed point numbers.

ICC is a counter for the number of multiply cycles which have been completed. NCC is the number of cycles needed.

The Multiplier Recode box is simulated as a block, to evaluate MY128X, MY64X, MY32X, MY16X, MY 8X, MY4X, MY2X, MY1X and NO, N1, N2, N3, N4. The equations for these signals are listed in the flowchart.

(5) The UM register holds the intermediate results. The output from the Recorder: MY128X, MY64X etc., used to determine how many bits the contents of M-register has to be left shifted and used as the Y-input to the SDS (Stage 1 through 4). NO is used to set NEGØ control for the negation of US. N1, N2, N3, N4 are used to set negation controls of the SDS (stages 1 through 4) which determine whether the multiple is added or subtracted. This is shown from point (4) to (5).

(6) The results from the 4th stage of SDS (T,Z) are right shifted 8 bits and transmitted into US and UM register. The contents of UQ register are also right shifted 8 bits.

(7) The multiply cycle is repeated from step (4), until NCC cycles have been completed.

(8) After NCC cycles have been completed, for long fixed point only, no right shifting will be performed. The contents of LS and LM are directly transmitted into US and UM registers respectively as at point (6).

(9) For floating point only, the routine MEXTPRE is called to assure the precision of the product.

(10) The result in US and UM registers is then assimilated by setting NEGØ and N(Negation control of SDS, Stage 1) and Y-input to 0. But in case of floating point operation and $UQ_{65} = 1$, one more addition is needed. This may be done by setting N = 1 and using the contents of the M register as Y-input, calling the 'ASSIM' routine . The assimilated result is put into the UQ register as shown for point (7) through (8).

(11) For floating point numbers, the routine 'NØRM' is used to normalize the product.

(12) In the case of short fixed point numbers, for negative products, the 4th byte of UQ must all be 1's. For positive products, the 4th byte of UQ must all be 0's, otherwise overflow occurs.

(13) In the case of long fixed point numbers, for negative products, byte 0-3 of UQ register must be all 1's, and for positive product they must all be 0's, otherwise overflow occurs.

(14) The routine 'SETBØG' is used to set BØGUS indicator and load FA with error indicators in case of overflow or underflow.

7.2 Flowchart:

The flowchart is attached.

Symbols used in the flowchart are:

SR: Sign of product (7 bit)

SIGNA: Sign of multiplier

SIGNB: Sign of multiplicand

NT: Number type

EUU: Exponent of multiplier

EUM: Exponent of multiplicand

EUL: Exponent of product

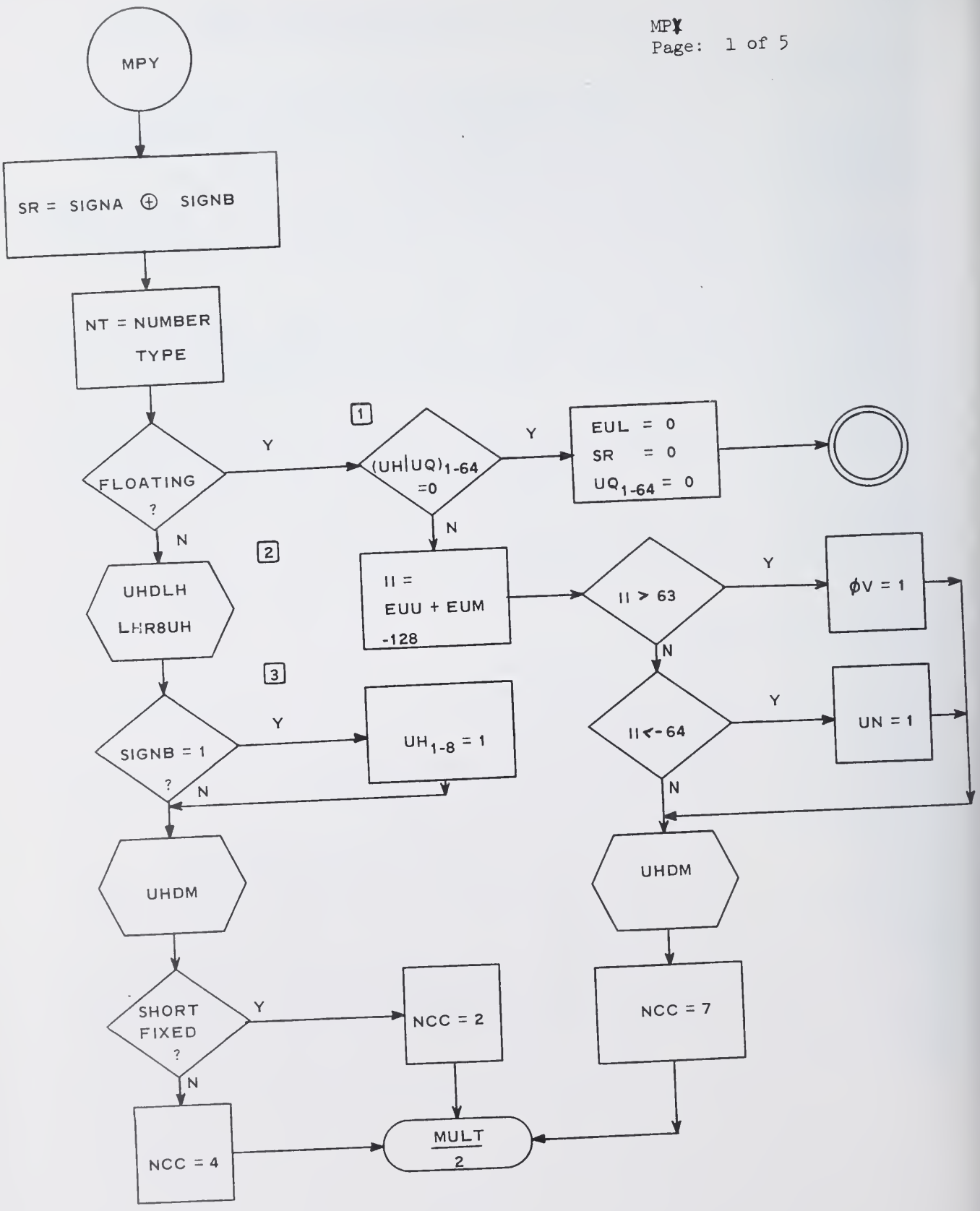
II: Extend EUL from 7 bit to 16 bit halfword.

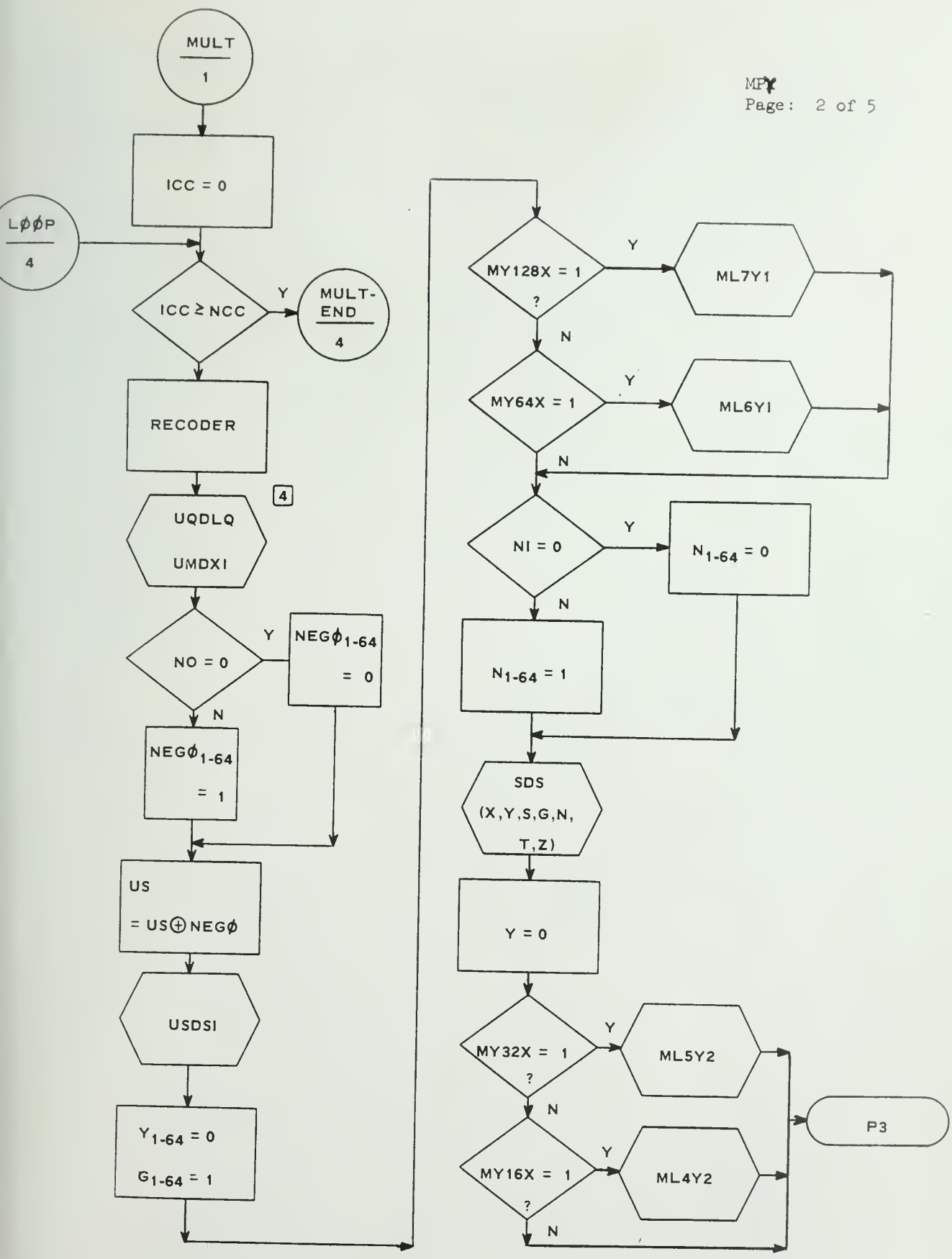
NCC: Number of multiply cycles needed.

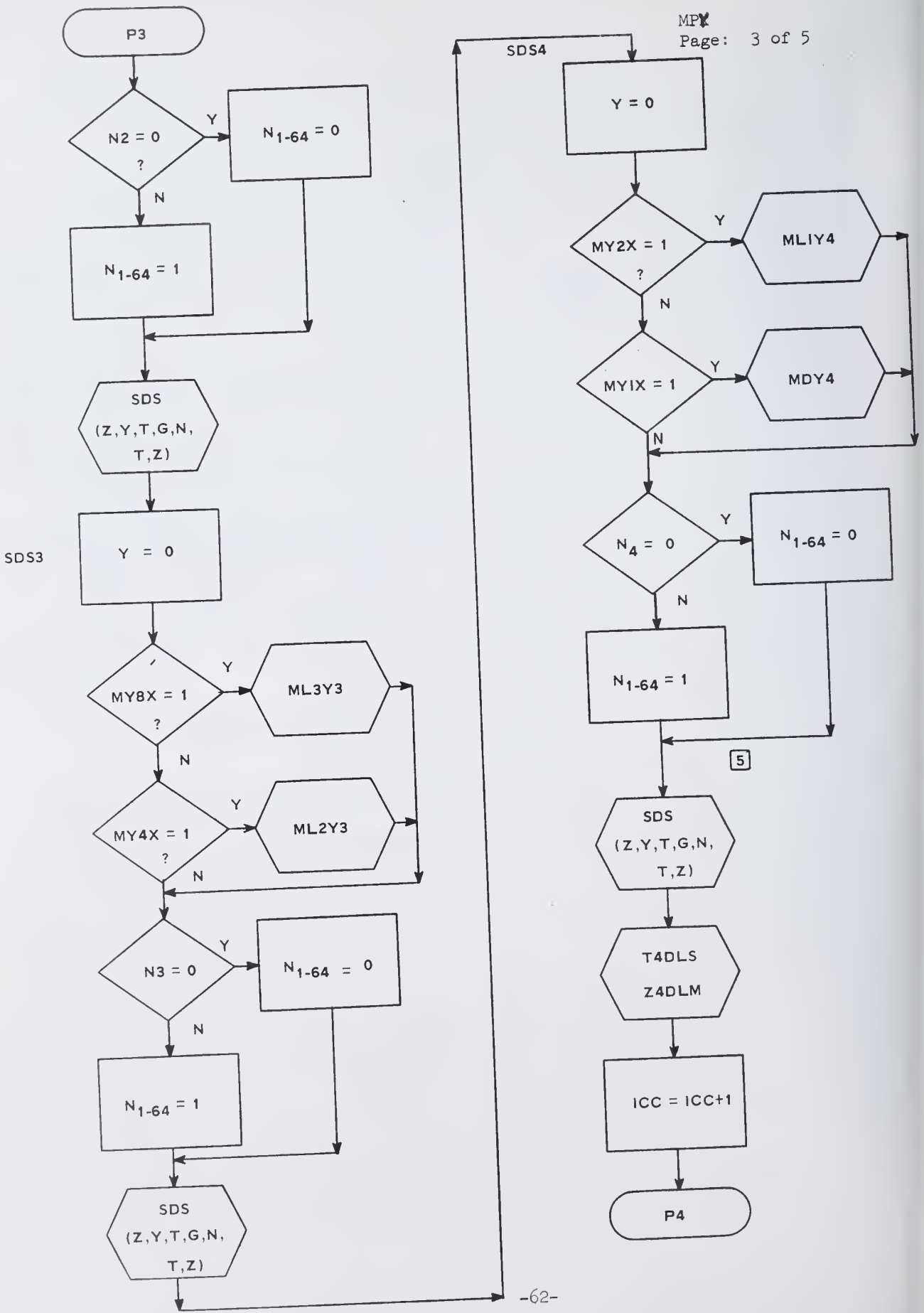
ICC: A counter for the multiply cycle being completed

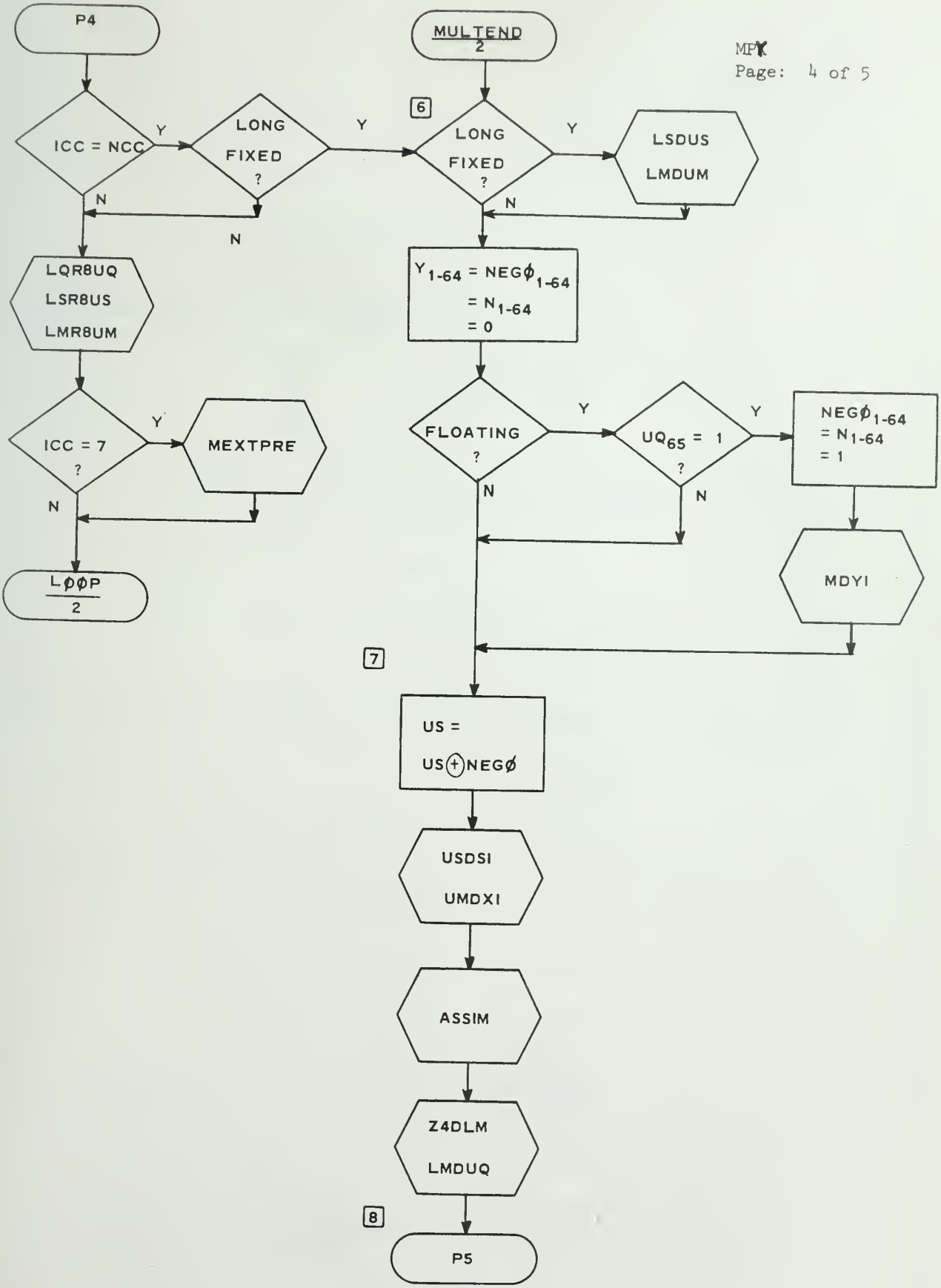
MY128X, MY64X, MY32X, MY16X, MY8X MY4X, MY2X, MY1X: Results from Multiplier Recoder to determine the shifting of the M register.

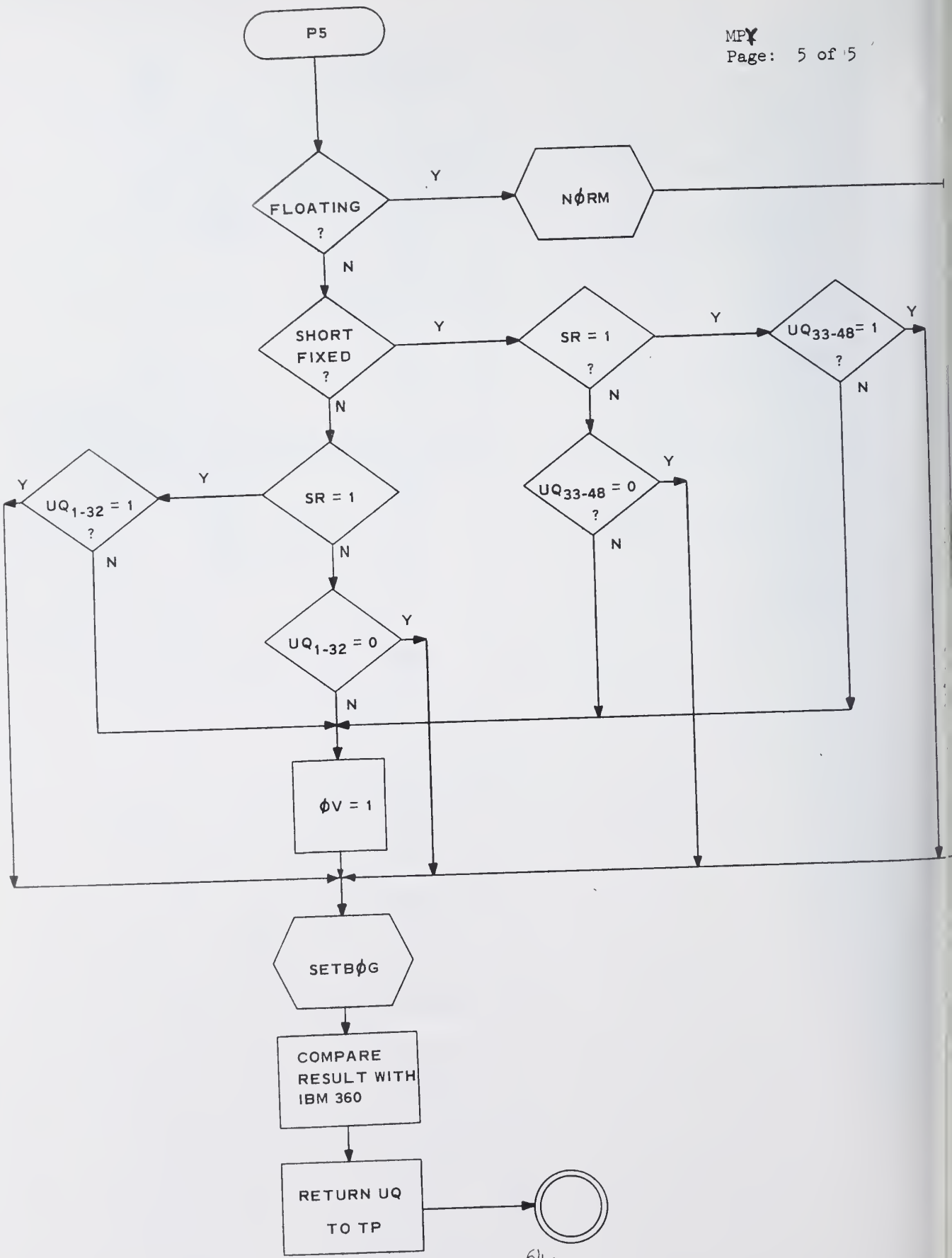
NO, N1, N2, N3, N4: Results from Multiplier Recoder to set the negation control for NEGØ and N's for 4 stages of SDS.

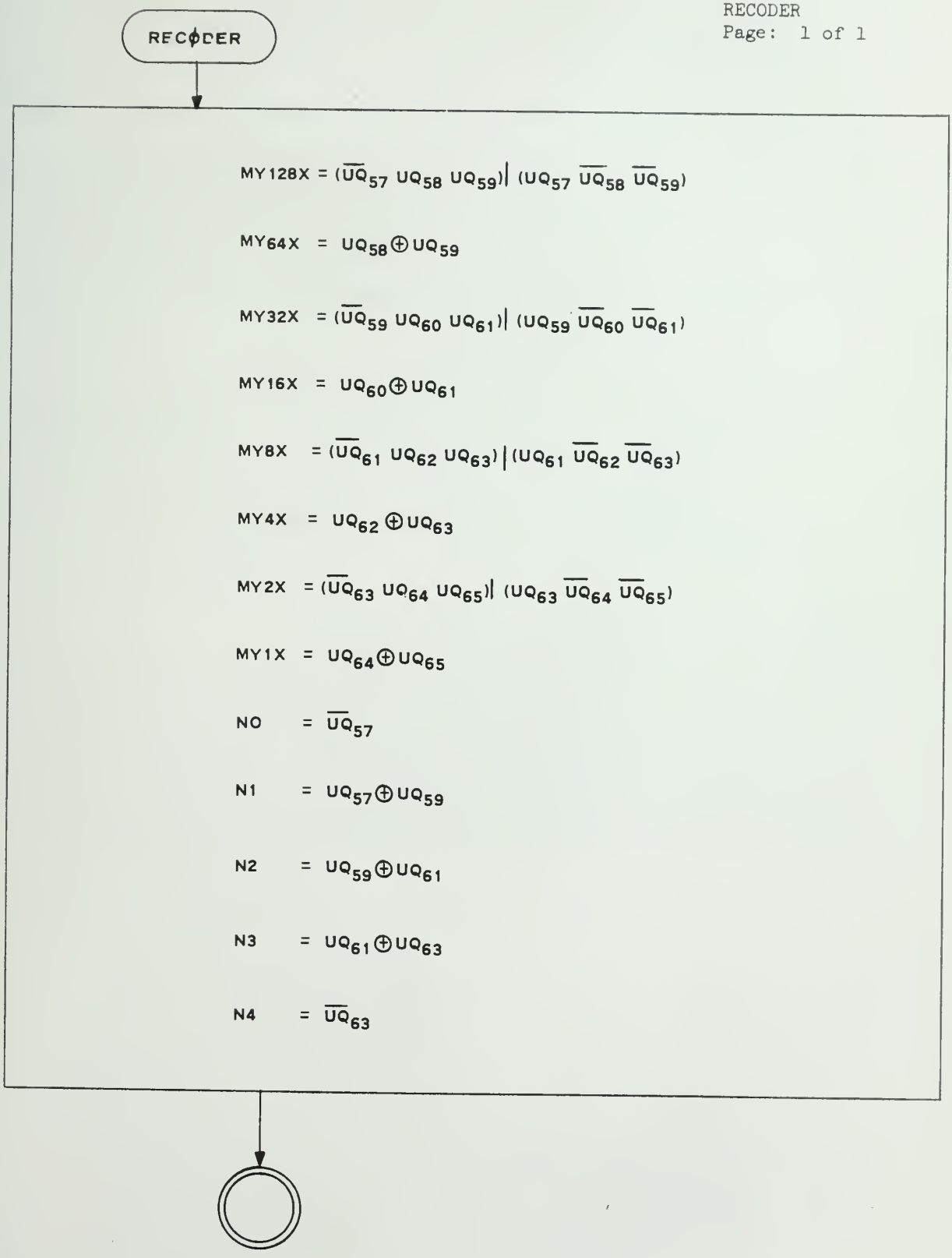

MPY
SR = SIGNA ⊕ SIGNB
NT = NUMBER TYPE
FLOATING ?
Y
N
1
(UH|UQ)1-64 =0
Y
EUL = 0
SR = 0
UQ1-64 = 0
N
2
UHDLH
LHR8UH
II = EUU + EUM -128
II > 63
Y
ϕV = 1
N
3
SIGNB = 1 ?
Y
UH1-8 = 1
N
II<-64
Y
UN = 1
N
UHDM
UHDM
SHORT FIXED ?
Y
NCC = 2
NCC = 7
N
NCC = 4
MULT
2


MULT
1
ICC = 0
LØØP
4
ICC ≥ NCC
Y
MULT-END
4
RECODER
4
UQDLQ
UMDX1
NO = 0
Y
NEGØ1-64 = 0
N
NEGØ1-64 = 1
US
= US⊕NEGØ
USDSI
Y1-64 = 0
G1-64 = 1
MY128X = 1
?
Y
ML7Y1
N
MY64X = 1
Y
ML6Y1
N
NI = 0
Y
N1-64 = 0
N
N1-64 = 1
SDS
(X,Y,S,G,N,
T,Z)
Y = 0
MY32X = 1
?
Y
ML5Y2
N
MY16X = 1
?
Y
ML4Y2
N
P3


P3
N2 = 0 ?
Y
N
N1-64 = 0
N1-64 = 1
SDS (Z,Y,T,G,N, T,Z)
SDS3
Y = 0
MY8X = 1 ?
ML3Y3
MY4X = 1 ?
ML2Y3
N3 = 0
N1-64 = 0
N1-64 = 1
SDS (Z,Y,T,G,N, T,Z)
SDS4
Y = 0
MY2X = 1 ?
MLIY4
MYIX = 1
MDY4
N4 = 0
N1-64 = 0
N1-64 = 1
5
SDS (Z,Y,T,G,N, T,Z)
T4DLS
Z4DLM
ICC = ICC+1
P4


P4
ICC = NCC
Y
LONG FIXED ?
Y
N
N
LQR8UQ
LSR8US
LMR8UM
ICC = 7 ?
Y
MEXTPRE
N
LØØP
2
MULTEND
2
6
LONG FIXED ?
Y
LSDUS
LMDUM
N
Y1-64 = NEGØ1-64
= N1-64
= 0
FLOATING ?
Y
UQ65 = 1 ?
Y
NEGØ1-64
= N1-64
= 1
N
N
MDYI
7
US =
US ⊕ NEGØ
USDSI
UMDXI
ASSIM
Z4DLM
LMDUQ
8
P5


P5
FLOATING ?
Y
NØRM
N
SHORT FIXED ?
Y
SR = 1 ?
Y
UQ33-48= 1 ?
Y
N
N
SR = 1
Y
UQ1-32 = 1 ?
Y
N
N
UQ33-48 = 0 ?
Y
N
UQ1-32 = 0
Y
N
ØV = 1
SETBØG
COMPARE RESULT WITH IBM 360
RETURN UQ TO TP


RECODER
MY128X = ($\overline{UQ}_{57}$ UQ_{58} UQ_{59}) | (UQ_{57} $\overline{UQ}_{58}$ $\overline{UQ}_{59}$)
MY64X = $UQ_{58} \oplus UQ_{59}$
MY32X = ($\overline{UQ}_{59}$ UQ_{60} UQ_{61}) | (UQ_{59} $\overline{UQ}_{60}$ $\overline{UQ}_{61}$)
MY16X = $UQ_{60} \oplus UQ_{61}$
MY8X = ($\overline{UQ}_{61}$ UQ_{62} UQ_{63}) | (UQ_{61} $\overline{UQ}_{62}$ $\overline{UQ}_{63}$)
MY4X = $UQ_{62} \oplus UQ_{63}$
MY2X = ($\overline{UQ}_{63}$ UQ_{64} UQ_{65}) | (UQ_{63} $\overline{UQ}_{64}$ $\overline{UQ}_{65}$)
MY1X = $UQ_{64} \oplus UQ_{65}$
NO = $\overline{UQ}_{57}$
N1 = $UQ_{57} \oplus UQ_{59}$
N2 = $UQ_{59} \oplus UQ_{61}$
N3 = $UQ_{61} \oplus UQ_{63}$
N4 = $\overline{UQ}_{63}$

X, Y, S: X, Y and S is put to SDS respectively. (Same name used for all 4 stages of SDS).

N: Negation control of SDS.(Same name used for 4 stages of SDS)

T, Z: T and Z output of SDS. (Same name used for all 4 stages of SDS).

NEGØ: Negation control of the S-input to SDS for the first stage of SDS only.

ASSIM: Assimilation routine.

SDS: Signed-digit-subtractor routine.

NØRM: Normalization routine.

SETBØG: Routine to set BØGUS indicator and FA.

MEXTPRE: A local procedure to improve the precision of product

FA: Normally holds the flag bits of the multiplier. In case of bogus result it holds error indicators.

MEPB: Multiplication extended precision buffer.

8. Error Conditions: None

9. Internal Procedures Defined in this routine:

MEXTPRE: For floating point multiplication only, at the end of the last multiply cycle, 4 bits are evaulated by the equations as listed in the flowchart. These 4 bits will replace the last 4 bits of fraction after normalization. So that the precision of the product remains to be 64 bits even after left shifting.

6.3 DIVISION

1. Name: DIV - Another entry name is CVDDIV, which is used by CVD only.

2. Function Description: This routine is used to perform either floating point division or fixed point division . For fixed point division if the divisor is zero, an error will occur. The ØV will be set, and the bogus result will be 2^{31}-1. For non-bogus cases the remainder is in the UQ register, bits 1-32 (with sign), and the quotient is in the UQ register, bits 33-64 (with sign).

   For floating point division, if the exponent of the quotient is greater than 63, or division by zero occurs, an overflow will occur. The bogus result will be the maximum floating point number that can be represented (with correct sign). If the exponent of the quotient is less than -64, an overflow occurs, the bogus result is the minimum floating point number that can be represented (with correct sign).

   A special use of this routine is defined by the name 'CVDDIV'. In this case, part of the fixed point division logic is used. The divisor is always 10 and only the remainder will be assimilated. The quotient remains in so-called SD format. The sign of the quotient will be the Exclusive ØR of the sign of the dividend and divisor. For fixed division the sign of the remainder is the same as the dividend.

3. Formal Calling Sequence - CALL DIV or CALL CVDDIV
4. Formal Parameter Description: None
5. Implicit Parameter Description: The UQ and UH registers are used to hold the dividend and divisor (the exact position will be described for each case in the following section). Since this routine will be used by CVD, there are two parameters which must be declared as external variables to communication between routines. They are NCC and NEGR. NCC is the number of divide cycles needed. NEGR is an indicator, which if set to 1, means the quotient must be decreased by 1.

6. Subroutines Used: ASSIM, LHL4UH, LHL8UH, LHR8UH, LMDM, LMDUM, LMDUQ, LML8UM, LQL4UQ, LQL8UQ, LQR4UQ, LQR8UQ, LSDUS, LSL8US, MDY1, MDY4, ML1Y4, ML2Y3, ML3Y3, ML4Y2, ML5Y2, ML6Y1, ML7Y1 NØRM, SDS, T4DLS, TIMGR, UHDLH, UHDM, UHDUS, UMDX1, UQDLQ, UQDUM, Z4DLM.

7. Operational Description:

7.1 English Text

1) For floating point division, the fraction of the dividend is taken from the UQ register, the exponent from EUU and the sign from SIGNA. The fraction of the divisor is taken from UH register, the exponent from EUM and the sign from SIGNB.

2) For fixed point division, the dividend is in the UQ register, bytes 0-3. The divisor is in both UQ register bytes 4-7 and UH register bytes 0-3. In this case, if both dividend and divisor are both positive numbers, no complementation is needed and the UQ register bytes 4-7 are cleared so that the dividend and divisor are in their proper position, UQ byte 0-3 and UH byte 0-3 respectively.

3) For the portion of the fixed divide logic shared by CVD (entry point CVDDIV), the dividend is in SDS format and loaded into UH/UQ registers. The divisor is always 10 and has built-in logic. The M-register is not used by the divide logic.

4) For floating point division, zero dividend and zero divisor will first be checked. No division cycle is needed. Otherwise the exponent of quotient will be the difference of EUU and EUM. The sign of quotient will be the exclusive ØR of SIGNA and SIGNB.

The divisor is loaded into M-register. The dividend is loaded into UM register. The partial quotient in SD format will be kept in UH/UQ register.

In the model division used in Illiac III, the range of the divisor is less than 1 and greater than or equal to 1/2. Therefore the M-register must be normalized to satisfy this requirement as at point 1. The number of division cycles needed for floating division is 7.

The routine 'DIVID' is used to perform the necessary division cycles. Since the quotient is in SD format, it must be assiminated into the conventional binary format by using the routine 'ASSIM'.

Since the divisor has been normalized (left shifted) into the range $1/2 \leq$ divisor <1, the quotient must be right shifted the same number of bits (as from point 2 to 3) before the normalization of the floating quotient by using 'NØRM'.

If any overflow or underflow occurs, the routine 'SETBØG' is used to set BØGUS and load FA.

Finally, the exponent of the quotient is in the UQ register, the exponent in EUL and the sign in SR. For bogus results, the error indicator is in FA, otherwise FA is unchanged.

5) For Fixed Point Division

For either negative dividend or negative divisor, the complement of UQ register is required, as in point 4 to 5. Before divide cycles begin, the dividend is in UQ register bytes 0-3, the divisor in UH register bytes 0-3.

In order to share the same divide logic as floating point division as well as to decrease the number of divide cycles needed, the UH and UQ registers are left (or right) shifted to fulfill the requirement. The actual number of division cycles needed will be one plus the difference of number of left shift 8 bits needed by the divisor and dividend.

As in floating division, the divisor (UH) is loaded into M; the dividend (UQ) is loaded into the UM register. UH/UQ registers are used to hold the quotient in SD form. The 'DIVIDE' routine is used to perform the divide cycles needed.

After the divide cycles are completed, the remainder is the T.Z output of SDS4 in the SDS format. It must be assiminated into conventional binary form. Since the quotient may be 1 greater than the actual quotient due to the model division used, the remainder must be added to the divisor to get to correct remainder if NEGR is set to 1. Since the remainder must be of the same sign as the dividend, for negative dividends, complementation is needed. In either case, another assimilation of quotient is needed.

Next the quotient must be assimilated (the remainder is moved to UQ register byte 0-3). If NEGR is set to 1, the quotient is decreased by 1 before assimilation as from points (6) to (7).

Since the dividend and divisor have been shifted to fulfill the requirements of the model division, so the remainder and quotient must be shifted to obtain the correct result.

Finally, the remainder and quotient are in the UQ register byte 0-3 and byte 4-7 respectively.

(6) For fixed division used by CVD

Starting at the entry CVDDIV, the NT is used to determine when to exit from the fixed division routine.

The only difference is that the dividend is in SD format and is taken from the UH/UQ registers. The M-register is not used by the 'DIVID' routine. The division is assumed to be 10, and a special set of shifting logic TENL7Y1, TENL6Y1, TENL5Y2, TENL4Y2, TENL3Y3, TENL2Y3, TENL1Y4 and TENDY4 are used instead of the M-shifting logic.

Only the remainder is to be assimilated; the quotient is in the SD format when exit from the CVDDIV routine occurs.

7.2 Flow chart

Flowchart is attached.

Symbols used in the flowchart are:

NCC: No. of division cycle needed (external variable)

ICC: A counter for the no. of division cycle have been completed.

II: EUL extended from 7-bit to 16-bit

DIVRL1: 1, if in floating division, M has been left shifted 1 bit

DIVRL2: = 1 " " " " 2 bits

DIVRL3: = 1, " " " " 3 bits

X,Y,S: X,Y, S input to SDS (same name used for all 4 stages of SDS)

N: Negation control for SDS (all 4 stages)

T, Z: T, Z output of SDS (all 4 stages)

NDRR8: = 1, if in fixed division, the UQ, UH registers have been right shifted 8 bits.

NDDL8: Number of times, UQ register has been left shifted 8 bits

NDRL8: " " " UH " " " " " " " "

NDRL4: Number of times, UH and UQ registers have been left shifted 4 bits.

NDRL1: Number of times, UH and UQ registers have been left shifted 1 bit.

NEGR: = 1, if quotient has to be subtracted by 1.

8. Error Condition:

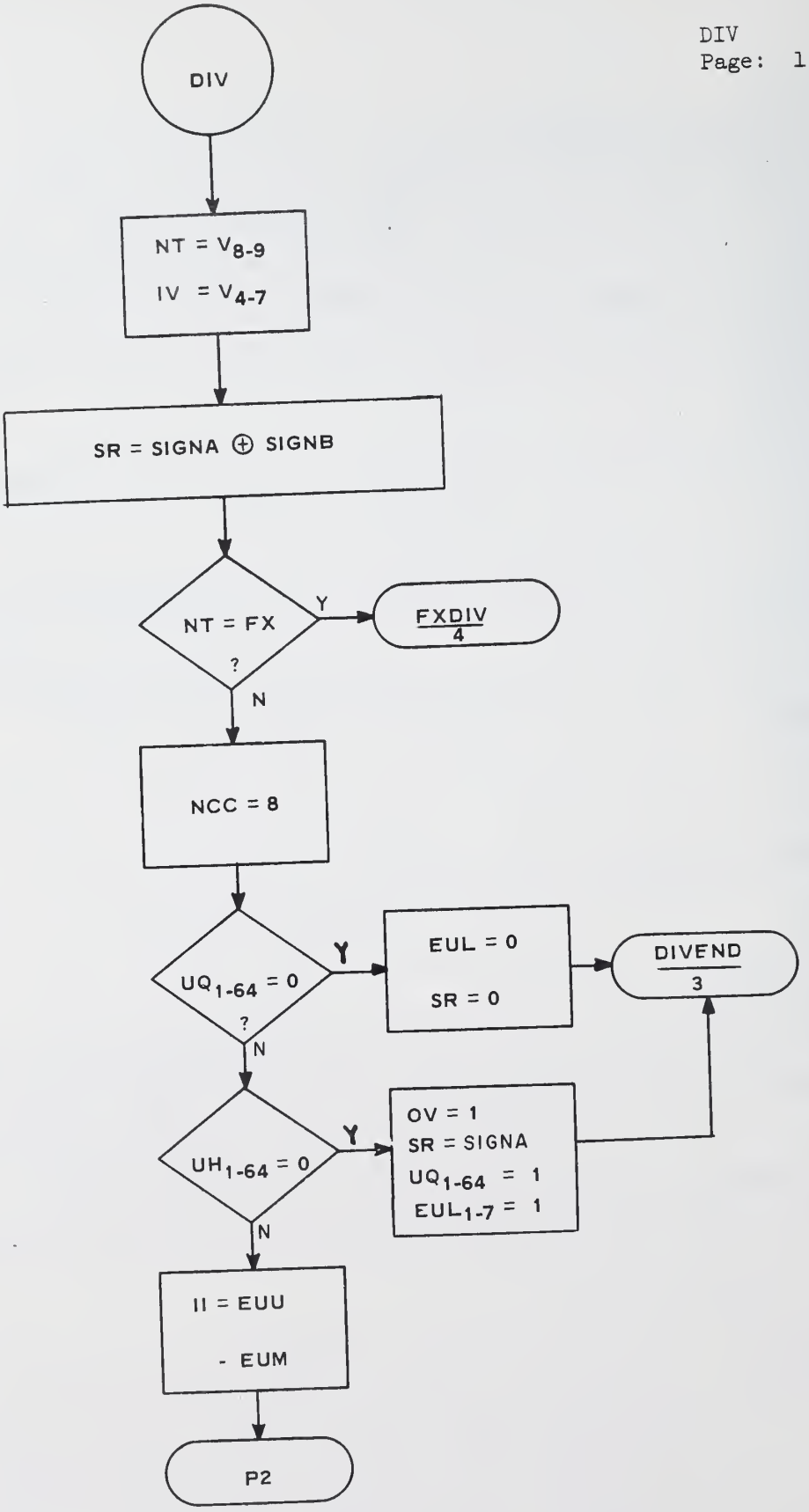
Floating Point Division: For overflow, bogus result is the maximum which can be represented (Sign of the result = Sign of the Dividend). For underflow, bogus result is the minimum number which can be represented (Sign of the Result = Sign of the Dividend).

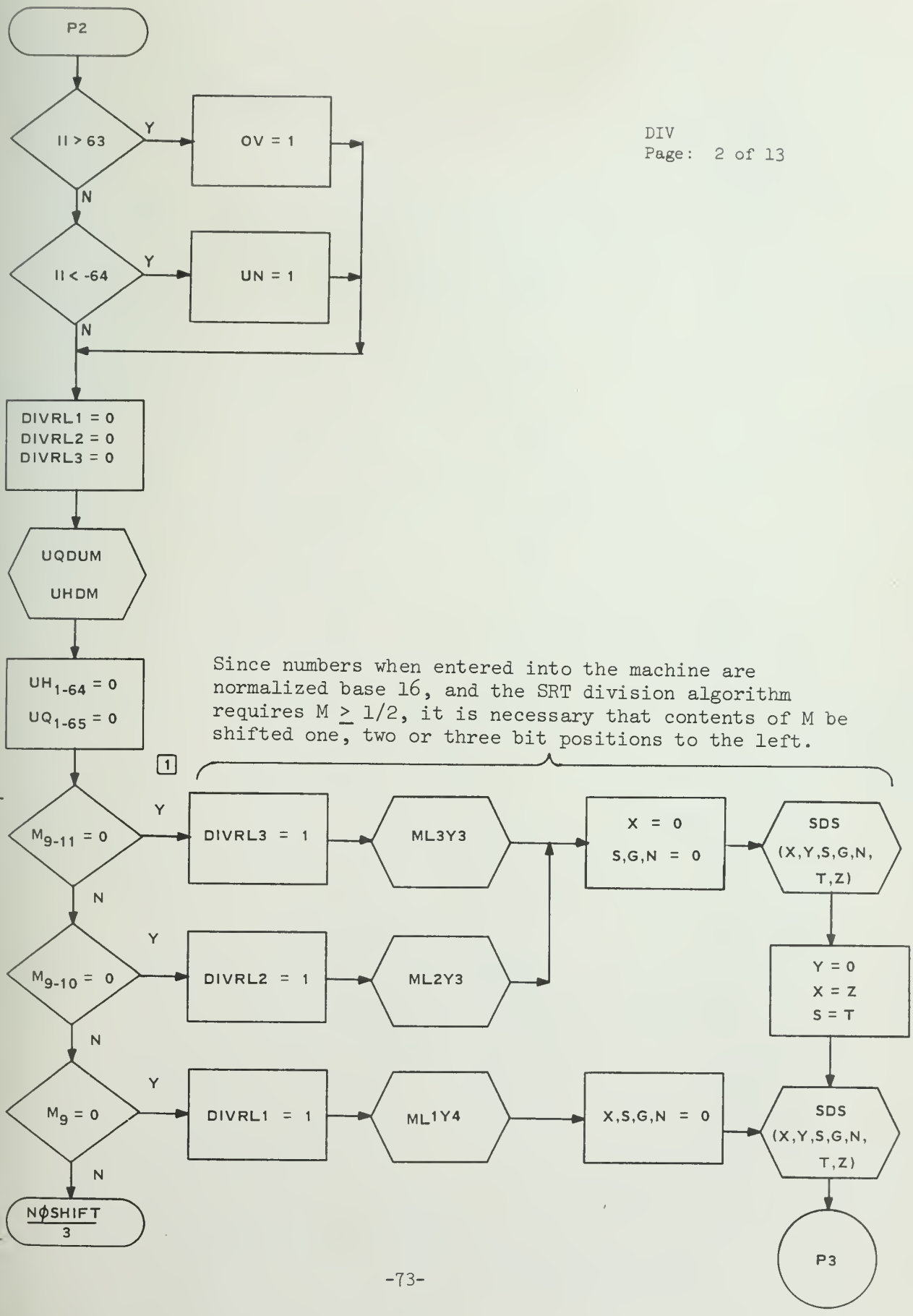
Fixed Point Division: For division by 0, the remainder will be 0, and quotient will be $2^{31}-1$.

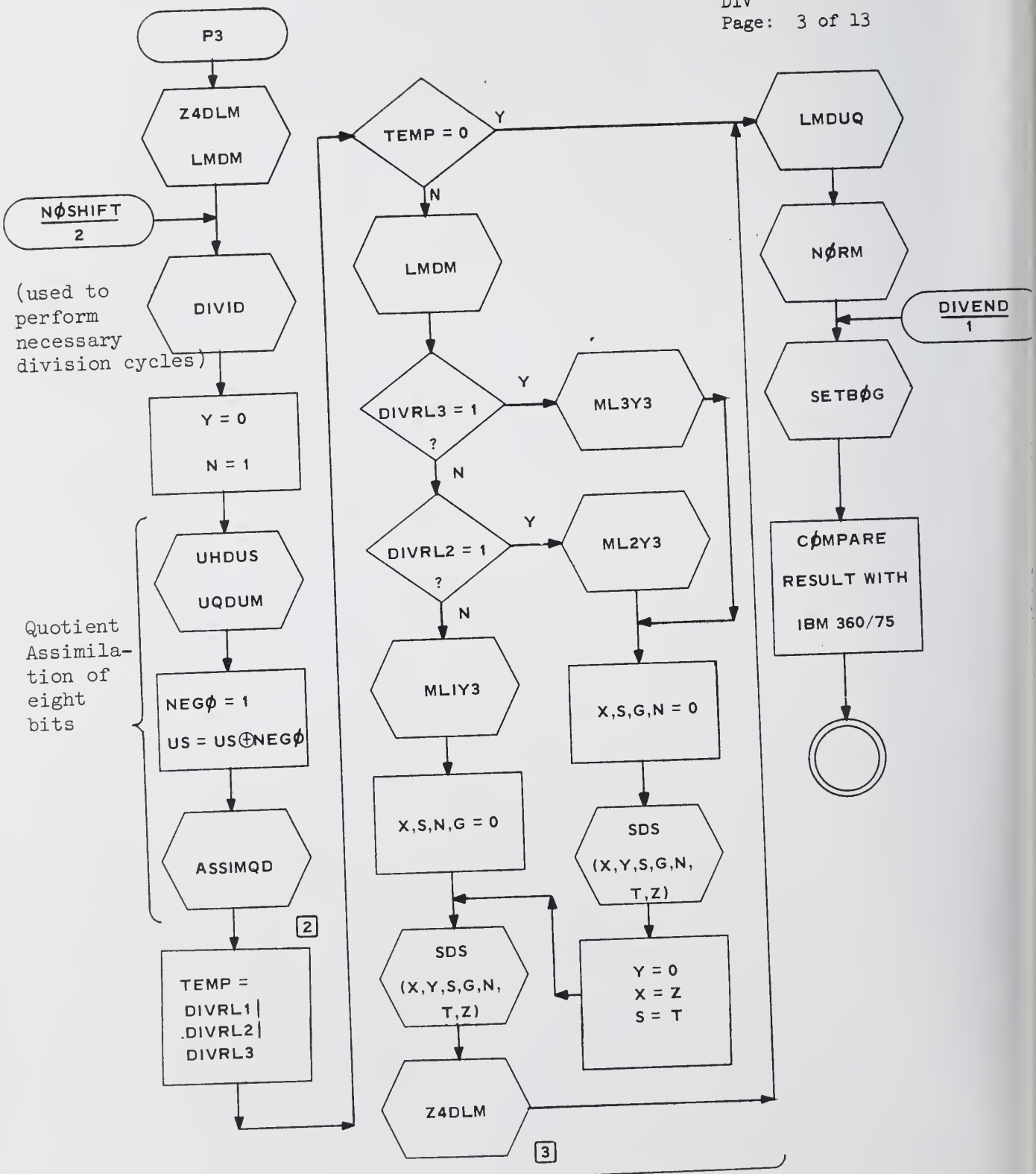
9. Internal Procedures Defined

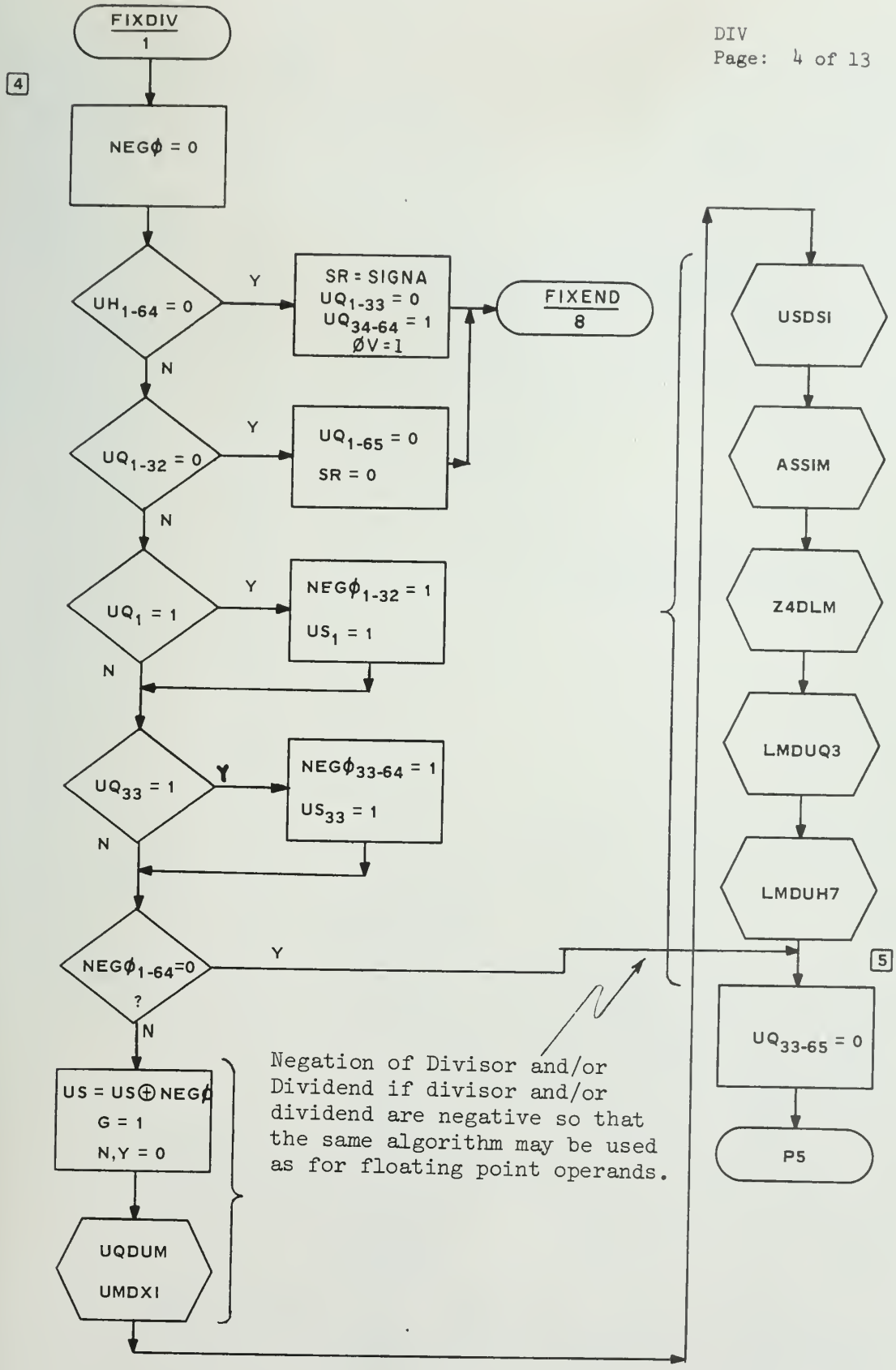
(i) ASSIMQD: This routine is used to assimilate

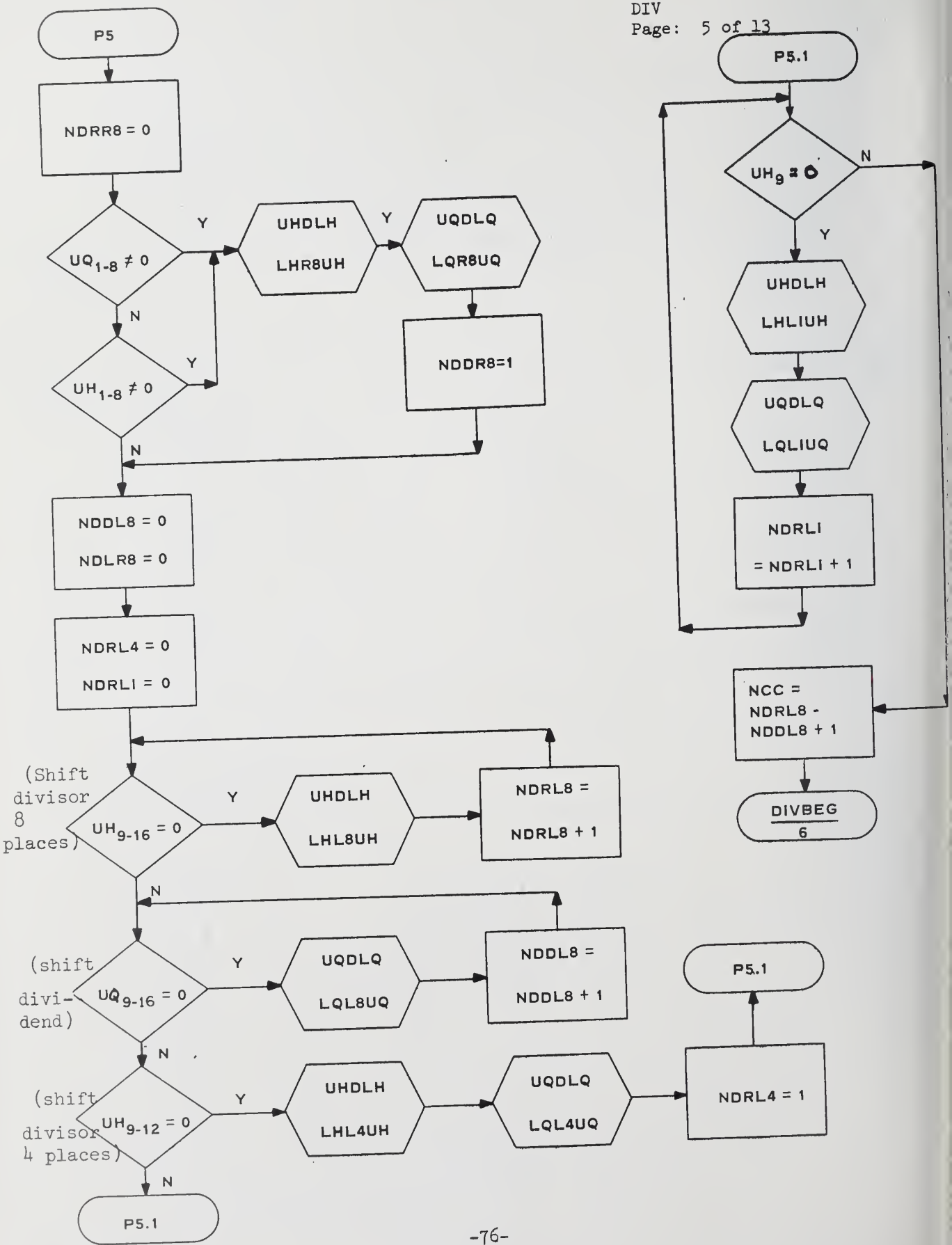
ASSIMER: the quotient (in register UH/UQ) from SD format to conventional binary form. The NEGØ, US, N and Y must be set before this routine is called. The flowchart is attached. The assimilated result is in the LM register.

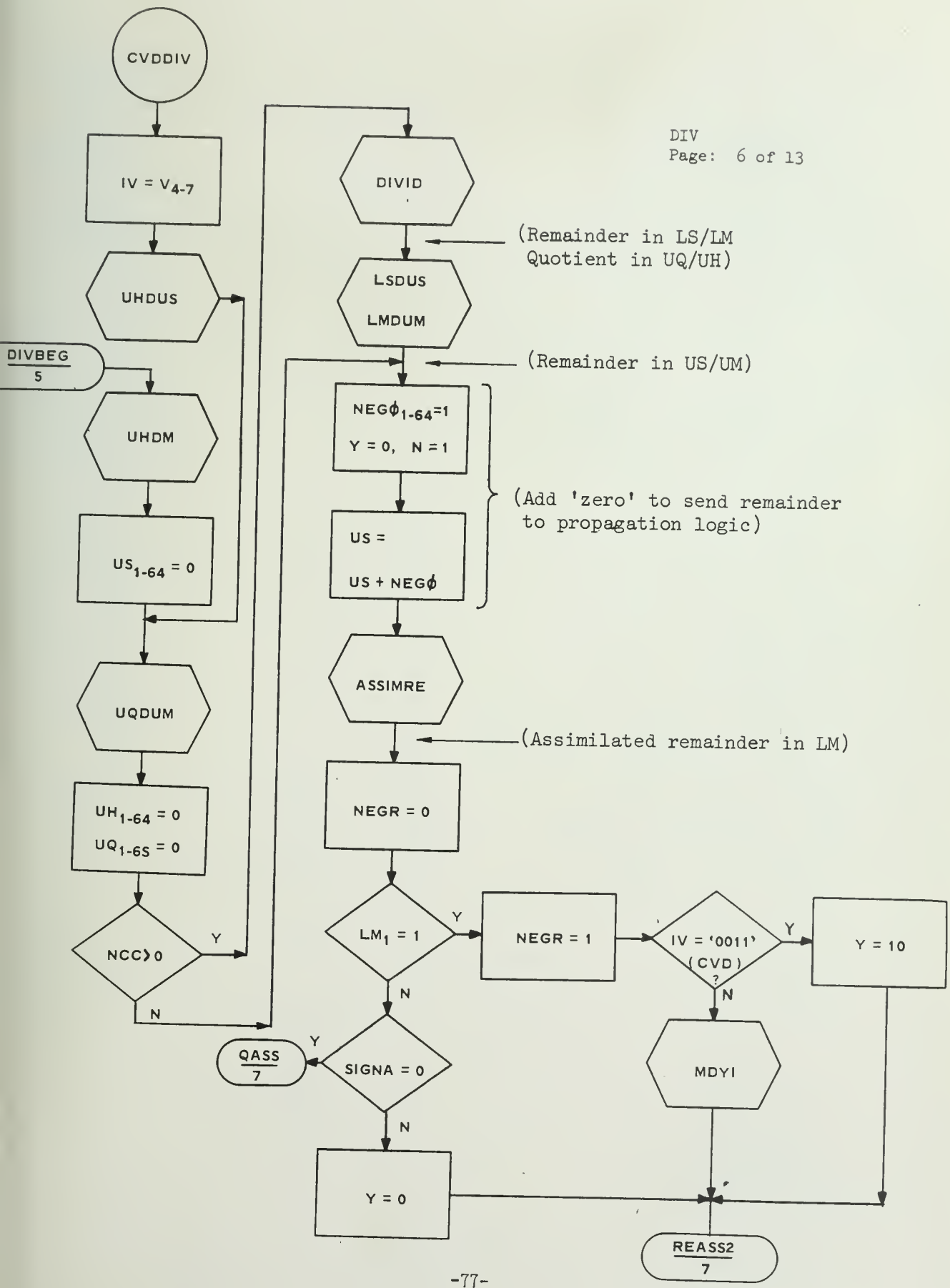

DIV
NT = V8-9
IV = V4-7
SR = SIGNA ⊕ SIGNB
NT = FX ?
Y
FXDIV / 4
N
NCC = 8
UQ1-64 = 0 ?
Y
EUL = 0
SR = 0
DIVEND / 3
N
UH1-64 = 0
Y
OV = 1
SR = SIGNA
UQ1-64 = 1
EUL1-7 = 1
N
II = EUU - EUM
P2

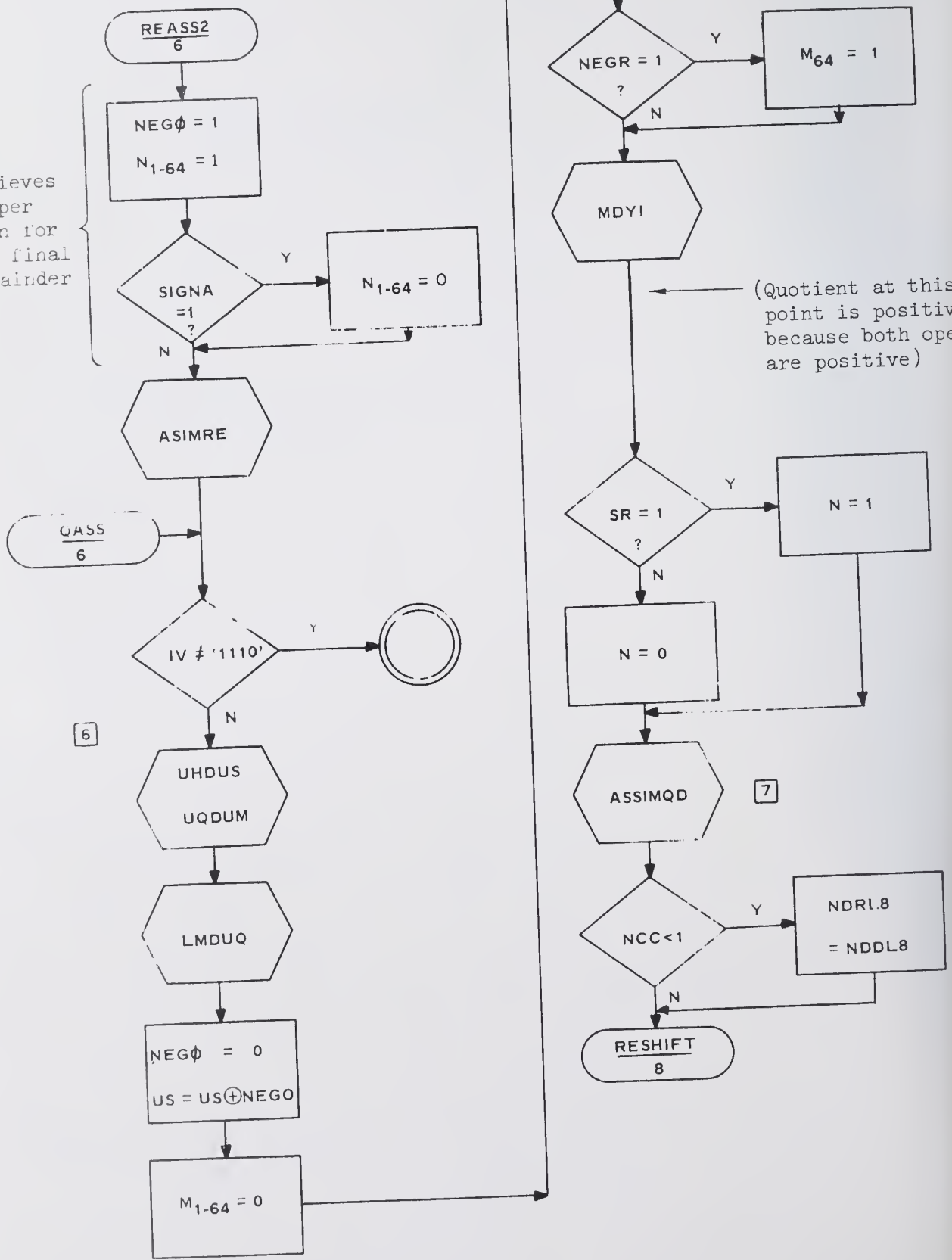

P2
II > 63
Y
OV = 1
N
II < -64
Y
UN = 1
N
DIVRL1 = 0
DIVRL2 = 0
DIVRL3 = 0
UQDUM
UHDM
UH1-64 = 0
UQ1-65 = 0
Since numbers when entered into the machine are normalized base 16, and the SRT division algorithm requires M ≥ 1/2, it is necessary that contents of M be shifted one, two or three bit positions to the left.
1
M9-11 = 0
Y
DIVRL3 = 1
ML3Y3
X = 0
S,G,N = 0
SDS
(X,Y,S,G,N,
T,Z)
N
M9-10 = 0
Y
DIVRL2 = 1
ML2Y3
Y = 0
X = Z
S = T
N
M9 = 0
Y
DIVRL1 = 1
ML1Y4
X,S,G,N = 0
SDS
(X,Y,S,G,N,
T,Z)
N
NØSHIFT
3
P3

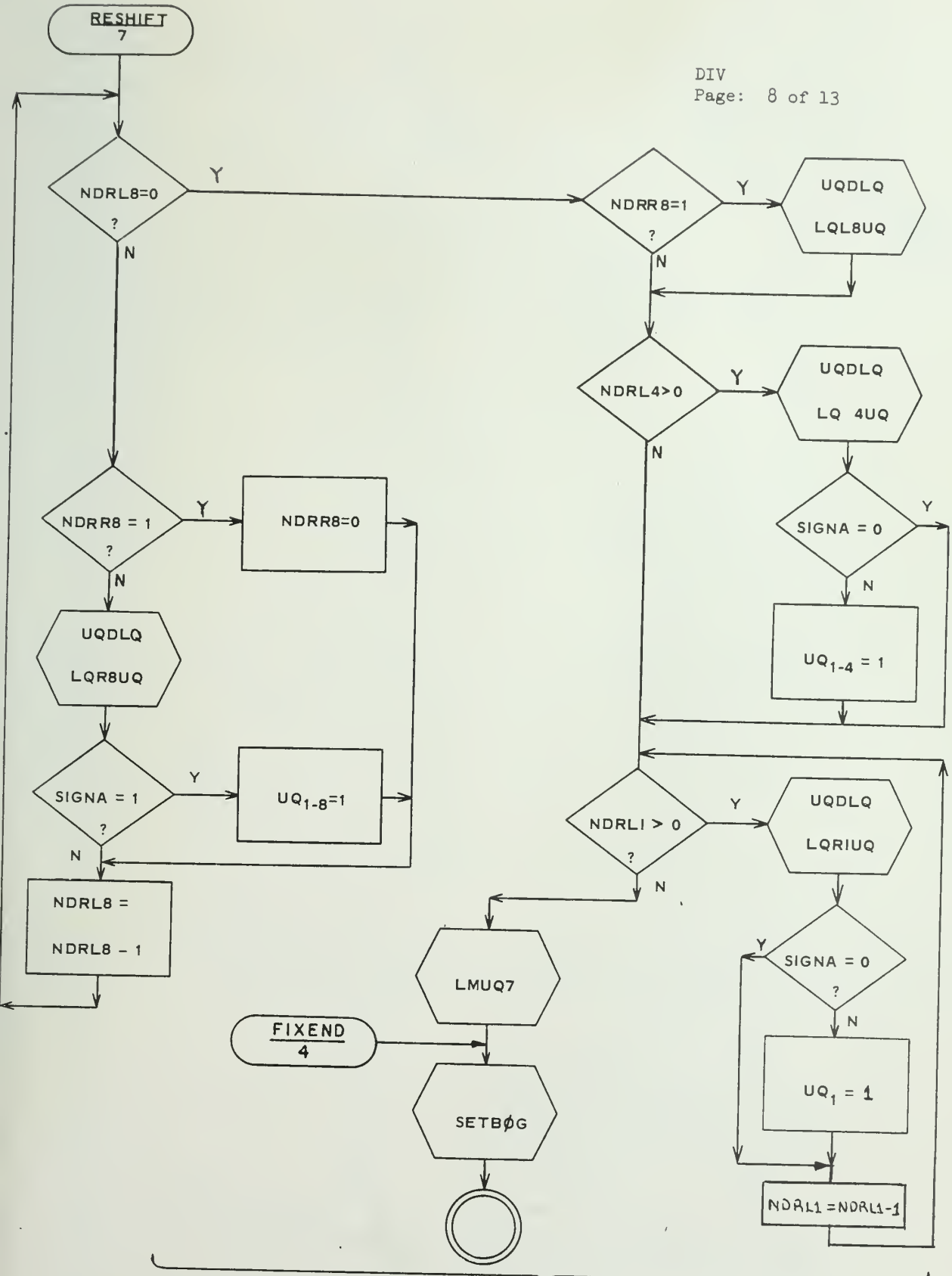

P3
Z4DLM
LMDM
NØSHIFT
2
(used to perform necessary division cycles)
DIVID
Y = 0
N = 1
Quotient Assimilation of eight bits
UHDUS
UQDUM
NEGØ = 1
US = US⊕NEGØ
ASSIMQD
2
TEMP = DIVRL1 | DIVRL2 | DIVRL3
TEMP = 0
Y
N
LMDM
DIVRL3 = 1 ?
Y
ML3Y3
N
DIVRL2 = 1 ?
Y
ML2Y3
N
MLIY3
X,S,N,G = 0
SDS (X,Y,S,G,N,T,Z)
SDS (X,Y,S,G,N,T,Z)
Z4DLM
3
X,S,G,N = 0
Y = 0
X = Z
S = T
LMDUQ
NØRM
DIVEND
1
SETBØG
COMPARE RESULT WITH IBM 360/75
Reshifting of quotient due to initial scaling of divisor M ≥ 1/2.


FIXDIV
1
4
NEGϕ = 0
UH1-64 = 0
Y
SR = SIGNA
UQ1-33 = 0
UQ34-64 = 1
ØV = 1
FIXEND
8
N
UQ1-32 = 0
Y
UQ1-65 = 0
SR = 0
N
UQ1 = 1
Y
NEGϕ1-32 = 1
US1 = 1
N
UQ33 = 1
Y
NEGϕ33-64 = 1
US33 = 1
N
NEGϕ1-64=0
?
Y
N
US = US⊕NEGϕ
G = 1
N,Y = 0
UQDUM
UMDXI
Negation of Divisor and/or Dividend if divisor and/or dividend are negative so that the same algorithm may be used as for floating point operands.
USDSI
ASSIM
Z4DLM
LMDUQ3
LMDUH7
5
UQ33-65 = 0
P5

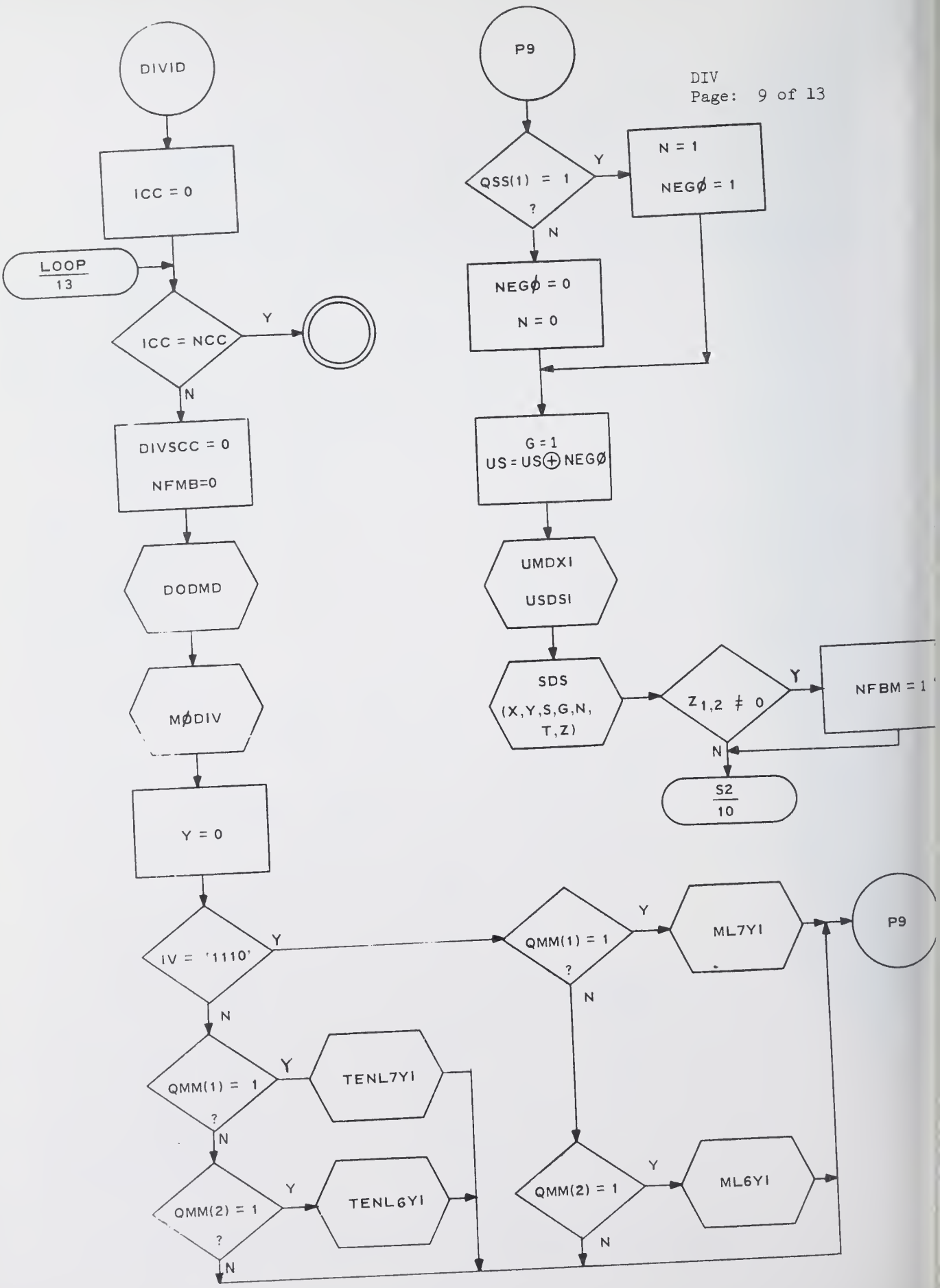

P5
NDRR8 = 0
UQ1-8 ≠ 0
Y
UHDLH
LHR8UH
Y
UQDLQ
LQR8UQ
N
UH1-8 ≠ 0
Y
NDDR8=1
N
NDDL8 = 0
NDLR8 = 0
NDRL4 = 0
NDRLI = 0
(Shift divisor 8 places)
UH9-16 = 0
Y
UHDLH
LHL8UH
NDRL8 = NDRL8 + 1
N
(shift divi-dend)
UQ9-16 = 0
Y
UQDLQ
LQL8UQ
NDDL8 = NDDL8 + 1
N
(shift divisor 4 places)
UH9-12 = 0
Y
UHDLH
LHL4UH
UQDLQ
LQL4UQ
NDRL4 = 1
P5.1
N
P5.1
P5.1
UH9 ≠ 0
N
Y
UHDLH
LHLIUH
UQDLQ
LQLIUQ
NDRLI = NDRLI + 1
NCC = NDRL8 - NDDL8 + 1
DIVBEG 6

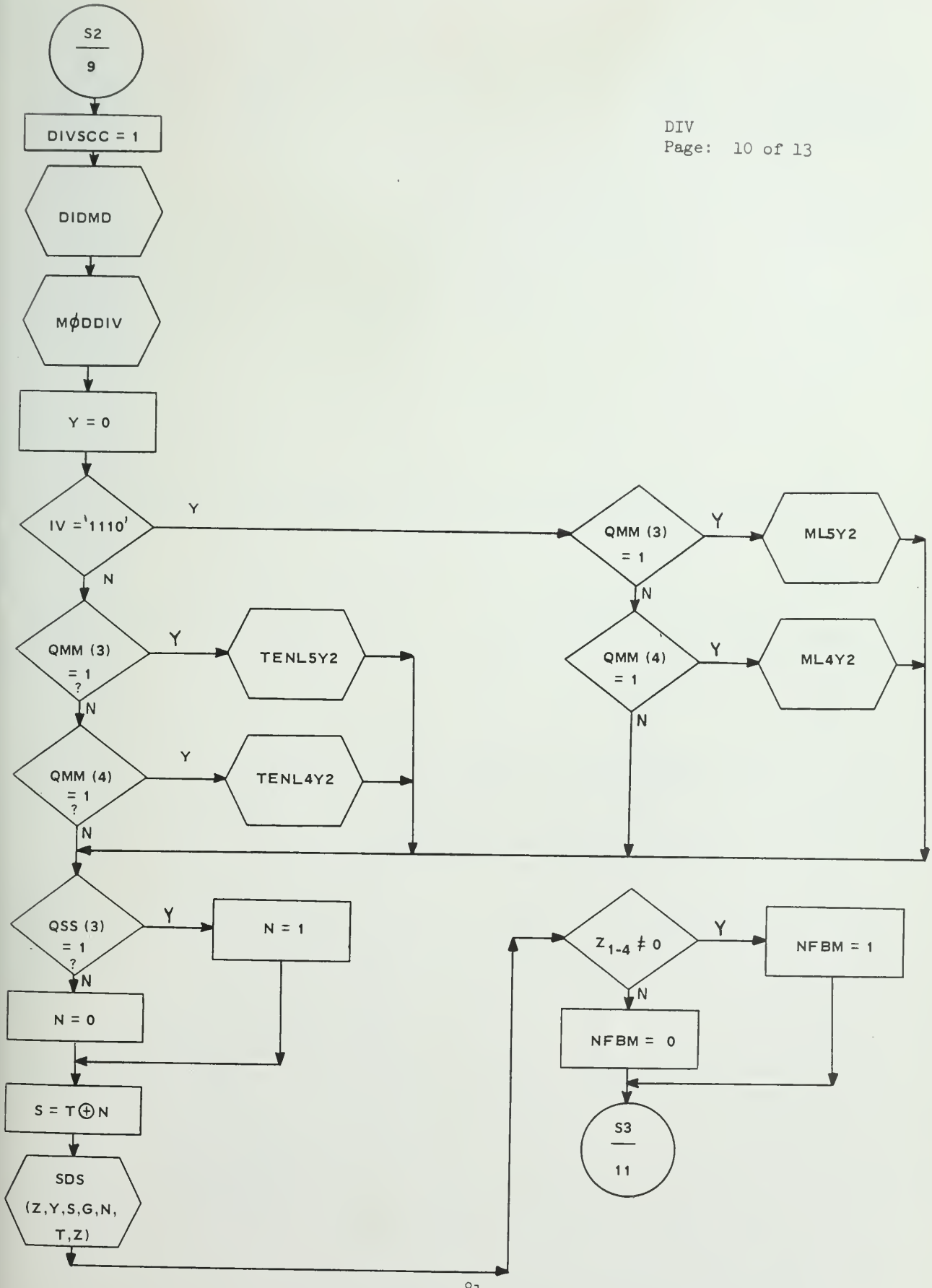

CVDDIV
IV = V4-7
UHDUS
DIVBEG 5
UHDM
US1-64 = 0
UQDUM
UH1-64 = 0
UQ1-6S = 0
NCC>0
Y
N
DIVID
(Remainder in LS/LM
Quotient in UQ/UH)
LSDUS
LMDUM
(Remainder in US/UM)
NEGφ1-64=1
Y = 0, N = 1
US =
US + NEGφ
(Add 'zero' to send remainder
to propagation logic)
ASSIMRE
(Assimilated remainder in LM)
NEGR = 0
LM1 = 1
NEGR = 1
IV = '0011' (CVD)?
Y = 10
MDYI
SIGNA = 0
QASS 7
Y = 0
REASS2 7

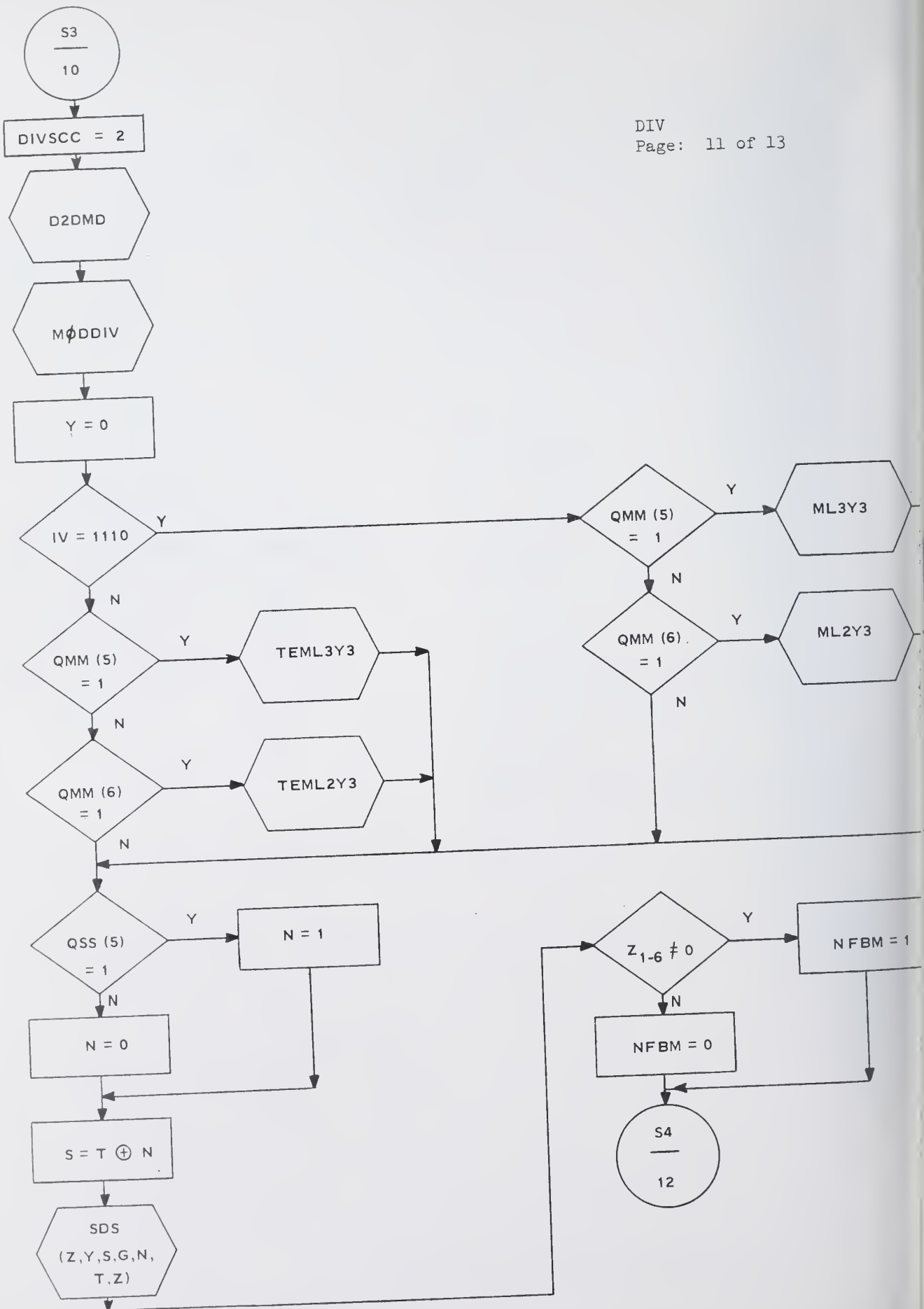

REASS2
6
NEGϕ = 1
N1-64 = 1
Achieves proper sign for the final remainder
SIGNA =1 ?
Y
N1-64 = 0
N
ASIMRE
QASS
6
IV ≠ '1110'
Y
N
6
UHDUS
UQDUM
LMDUQ
NEGϕ = 0
US = US⊕NEGO
M1-64 = 0
NEGR = 1 ?
Y
M64 = 1
N
MDYI
(Quotient at this point is positive because both operand are positive)
SR = 1 ?
Y
N = 1
N
N = 0
ASSIMQD
7
NCC<1
Y
NDRI.8 = NDDL8
N
RESHIFT
8

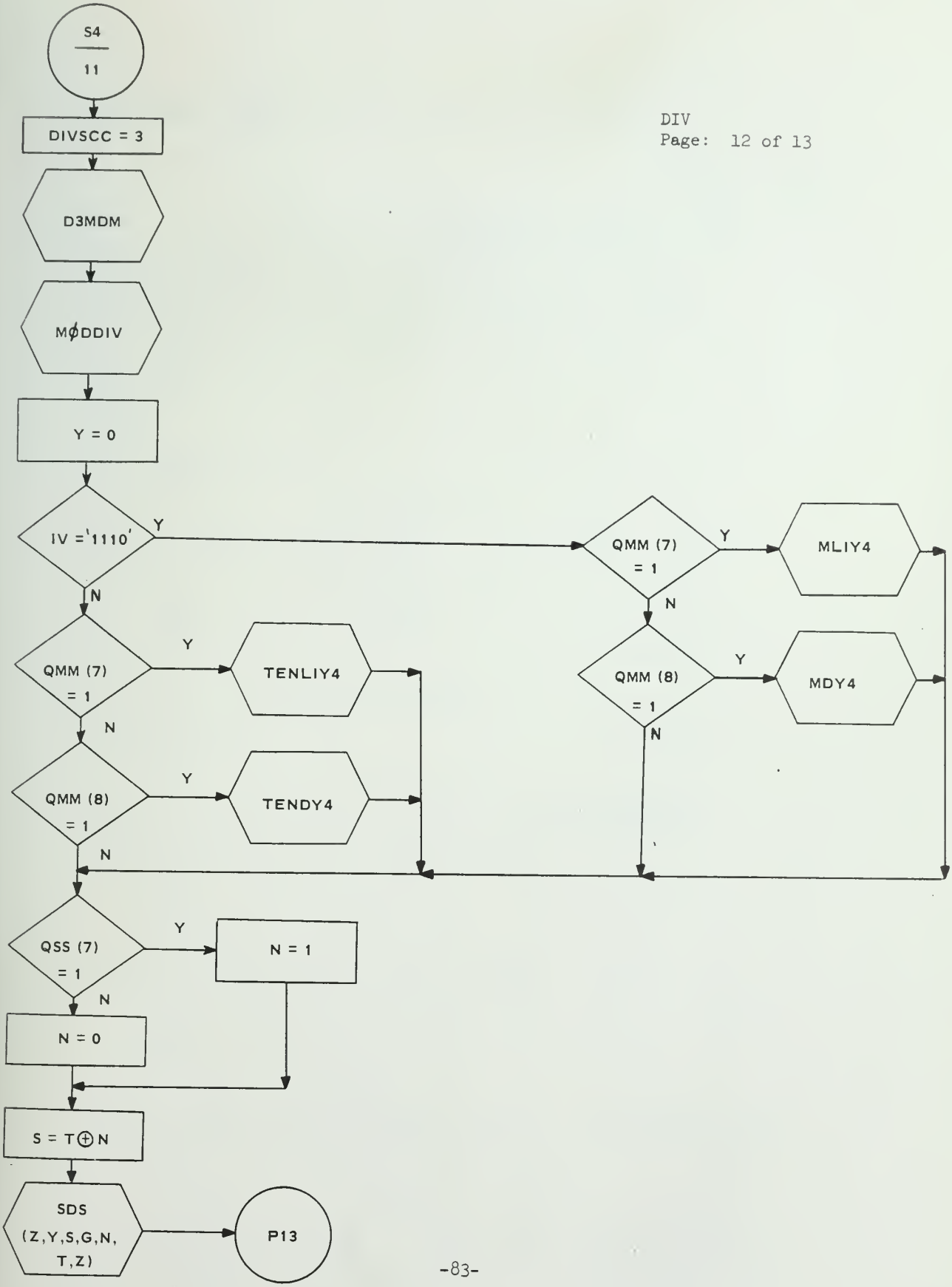


(Post shifting of Remainder to correct for initial scaling of Divisor)

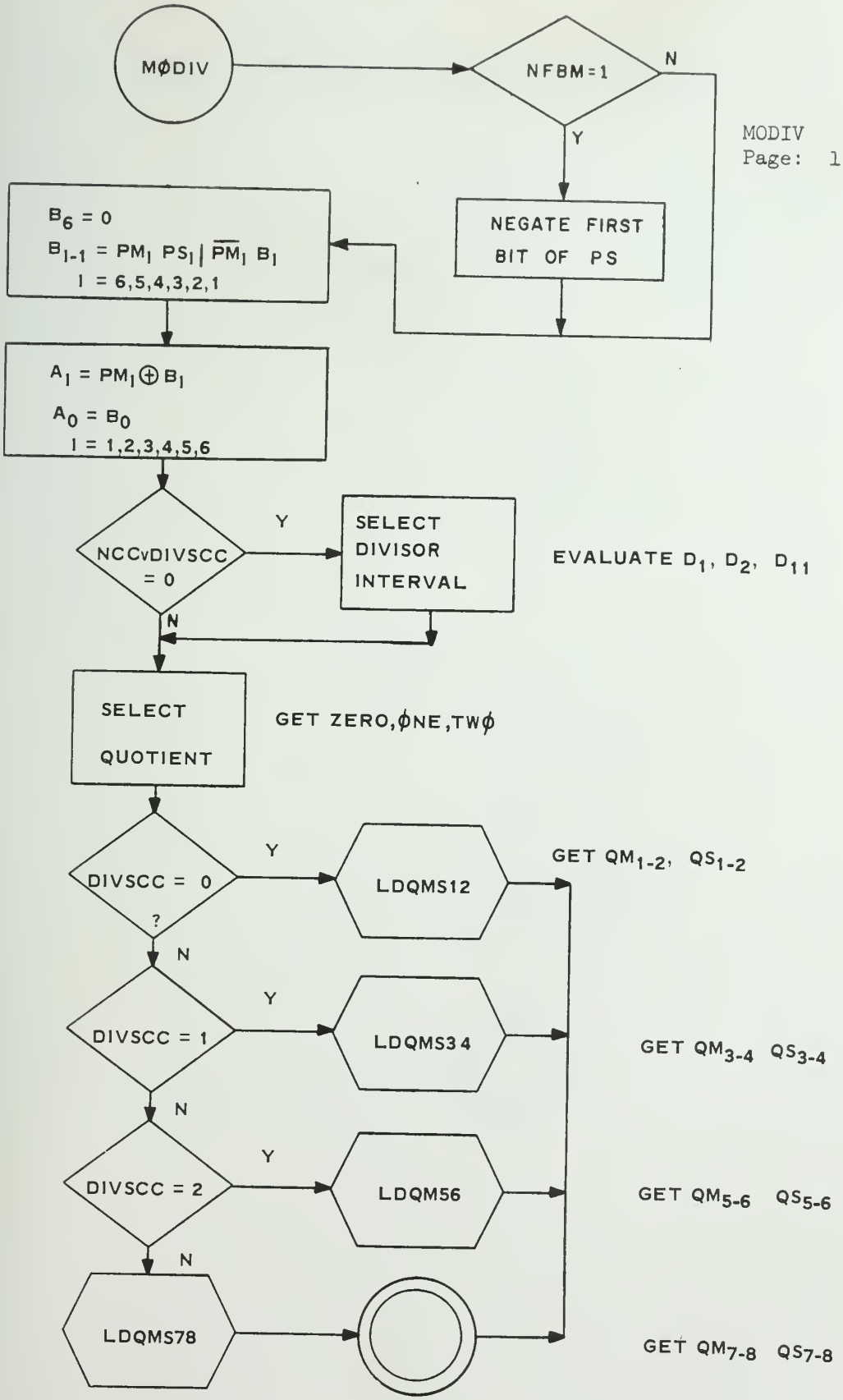

DIVID
ICC = 0
LOOP
13
ICC = NCC
Y
N
DIVSCC = 0
NFMB=0
DODMD
MØDIV
Y = 0
IV = '1110'
Y
N
QMM(1) = 1
?
Y
N
TENL7YI
QMM(2) = 1
?
Y
N
TENL6YI
P9
QSS(1) = 1
?
Y
N
N = 1
NEGØ = 1
NEGØ = 0
N = 0
G = 1
US = US⊕NEGØ
UMDXI
USDSI
SDS
(X,Y,S,G,N,
T,Z)
Z1,2 ≠ 0
Y
N
NFBM = 1
S2
10
QMM(1) = 1
?
Y
N
ML7YI
P9
QMM(2) = 1
Y
N
ML6YI

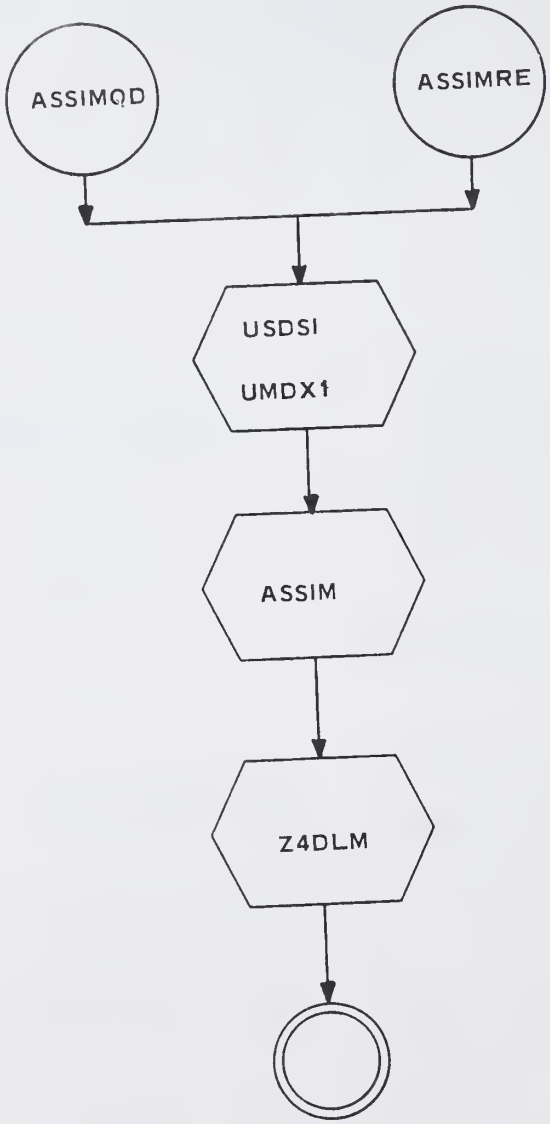

S2
9
DIVSCC = 1
DIDMD
MØDDIV
Y = 0
IV = '1110'
Y
N
QMM (3) = 1 ?
Y
TENL5Y2
N
QMM (4) = 1 ?
Y
TENL4Y2
N
QMM (3) = 1
Y
ML5Y2
N
QMM (4) = 1
Y
ML4Y2
N
QSS (3) = 1 ?
Y
N = 1
N
N = 0
S = T⊕N
SDS
(Z,Y,S,G,N,
T,Z)
Z1-4 ≠ 0
Y
NFBM = 1
N
NFBM = 0
S3
11


S3
10
DIVSCC = 2
D2DMD
MØDDIV
Y = 0
IV = 1110
Y
N
QMM (5) = 1
Y
TEML3Y3
N
QMM (6) = 1
Y
TEML2Y3
N
QMM (5) = 1
Y
ML3Y3
N
QMM (6) = 1
Y
ML2Y3
N
QSS (5) = 1
Y
N = 1
N
N = 0
S = T ⊕ N
SDS
(Z,Y,S,G,N,
T,Z)
Z1-6 ≠ 0
Y
NFBM = 1
N
NFBM = 0
S4
12


S4
11
DIVSCC = 3
D3MDM
MØDDIV
Y = 0
IV ='1110'
Y
N
QMM (7) = 1
Y
MLIY4
N
QMM (8) = 1
Y
MDY4
N
QMM (7) = 1
Y
TENLIY4
N
QMM (8) = 1
Y
TENDY4
N
QSS (7) = 1
Y
N = 1
N
N = 0
S = T⊕N
SDS (Z,Y,S,G,N, T,Z)
P13


P13
ICC ≠ NCC-1
Y
N
$Z_{1-8} \neq 0$
Y
N
$T_9 = \overline{T}_9$
T4DLS
Z4DLM
LSL8US
LML8UM
LHDLH
LHL8UH
UQDLQ
LQL8UQ
QBDUQH
ICC = ICC + 1
LØØP
9


MØDIV
NFBM=1
N
Y
MODIV
Page: 1 of 1
NEGATE FIRST BIT OF PS
$B_6 = 0$
$B_{I-1} = PM_I\ PS_I \mid \overline{PM}_I\ B_I$
$I = 6,5,4,3,2,1$
$A_I = PM_I \oplus B_I$
$A_0 = B_0$
$I = 1,2,3,4,5,6$
NCCvDIVSCC = 0
Y
SELECT DIVISOR INTERVAL
EVALUATE D_1, D_2, D_{11}
N
SELECT QUOTIENT
GET ZERO,ØNE,TWØ
DIVSCC = 0 ?
Y
LDQMS12
GET QM_{1-2}, QS_{1-2}
N
DIVSCC = 1
Y
LDQMS34
GET QM_{3-4} QS_{3-4}
N
DIVSCC = 2
Y
LDQM56
GET QM_{5-6} QS_{5-6}
N
LDQMS78
GET QM_{7-8} QS_{7-8}


ASSIMOD
ASSIMRE
USDSI
UMDX1
ASSIM
Z4DLM

(ii) LMDUQ3: This routine is used by fixed division

$UQ_{1-32} = LM_{1-32}$

(iii) LMDUH7: This routine is used by fixed division $UH_{1-32} = LM_{1-32}$

(iv) LQL1UQ: This routine is used by fixed division

$UQ_{1-64} = LQ_{2-64}$ $UQ_{65=0}$

(v) LQR1UQ: Used by fixed division

$UQ_1 = 0$

$UQ_{2-64} = LQ_{1-64}$

(vi) LHL1UH: Used by fixed division

$UH_{1-63} = LH_{2-64}$

$UH_{64} = 0$

(vii) LMUQ7: $UQ_{33-64} = LM_{33-64}$

(viii) Shifting logic used by CVD DIV only

TENL7Y1: Y_2, Y_4 = 1, other bits of Y = 0

TENL6Y1: Y_3, Y_5 = 1, " " "

TENDY1: Y_9, Y_{11} = 1, " " "

TENL5Y2: Y_4, Y_6 = 1, " " "

TENL4Y2: Y_5, Y_1 = 1, " " "

TENL3Y3: Y_6, Y_8 = 1, " " "

TENL2Y3: Y_7, Y_9 = 1, " " "

TENL1Y4: Y_8, Y_{10} = 1, " " "

TENDY4: Y_9, Y_{11} = 1, " " "

(ix) DIVID

This routine is used to perform NCC division cycles. Each division cycle requires stepping through 4 stages of SDS.

(a) The routines DODMD, D1DMD, D2DMD, D3DM4 are used to get the 6-bits which will be used by the MØDIV for each stage of SDS,

(b) Then MØDIV (Model Division) is called.

(c) The output for Model Division:

$QM_{1,2}$ are used to determine the shifting of M register in Stage 1 of SDS (or TEN shift for CVD)

$QM_{3,4}$ for stage 2 of SDS

$QM_{5,6}$ " 3

$QM_{7,8}$ for stage 4 of SDS

QS_1 are used to set the negation control of SDS1.

QS_3 " " " SDS2

QS_5 " " " SDS3.

QS_7 " " " SDS4.

(d) For each step in a cycle (except in the last SDS of the last cycle of fixed division), the overflow due to SDS is detected, and the indicator NFBM is set to 1. Then in the MØDIV, if NFBM = 1 the first bit of PS will be negated to correct the overflow.

(3) For the routine 'MØDIV', flowchart is attached.

PM, PS: Output from DODMD
D1DMD
D2DMD
D3DMD

DIVSCC: No. of stage of SDS (0-3).

NCC: No. of divides cycles needed.

LDQMS12
LDQMS34
LDQMS56
LDQMS78
} to get QM_{1-8} QS_{1-8}

6.4 Data Conversion

All routines used for data conversions are grouped in one routine, 'CØNVS'. There are three types of data conversion: convert to long fixed point number (CVL), convert to floating point number (CVF), and convert to decimal number (CVD). These are described in the following sections. Also three auxilary subroutines: 'DVB', which is used to convert decimal digits into binary bits; 'CØMPL', which is used to algebraicly complement the contents of the QUQ register; and 'FL-NØRL-FX', which is used to align the floating point no. into double word fixed point form will be described. Since some subfunctions are common to CVL, CVF, and CVD, one overall flowchart for data conversion is provided. Reference points used in the following section is marked by N in the flowchart.

Page 1-3 of the flowchart are for CVL. Pages 4-5 for CVF, pages 6-9 for CVD, pages 10-11 for DVB, page 12 for CØMPL, and page 13 for FL-NØRL-FX.

The symbols used in this chart are:

V: INBUS (50 bits)

NT: Number type (2 bits)

DEC: Decimal (for NT = '11')

FL: Floating (for NT ='10')

IØPI: A storage buffer to store the converted number in conventional binary form.

DØPI: A buffer to store the number to be converted in IBM 360/75 decimal form.

FØPI: A buffer to store the number to be converted in floating point form.

DVB: A routine to convert decimal digits (in UQ) to binary bits

ASSIM: A routine to convert SDS format to conventional binary form. For detail see the write up 'GATE'.

II: Extends the exponent of a floating point number from 7 bits to a half word integer.

SETBØG: A routine to check ØV, UN, LSG or ID set up BØGUS indicator and FA. For more detail see the write-up for 'GATE'.

ITEMP: Temporary storage.

NDDL8: Temporary storage to store the number of times UQ and UH registers being left-shifted 8 bits.

ICC: A temporary count.

UH, UQ, LH, LQ, US, LS, UM, LM: Registers (64 bit)

FA: Flag register (8 bits)

SIGNA, SIGN B, SR: Sign Register (1 bit)

ØV, UN: Overflow, Underflow indicators.

X, Y, S: Inputs to the signed-digit-subtractor (same name used for all 4 stages of SDS)

NEGØ: Negation control for S-input to SDS (Stage 1).

N: Negation control for T-output of SDS. (Same name used for all 4 stages of SDS).

T,Z: Outputs of the SDS (all 4 stages).

CVDDIV: A routine to perform the division (by 10). Used by CVD only. For detail see the write-up for 'DIV'.

LHL4UH, LHL8UH, LMDM, LMDUM, LMDUQ, LQL4EQ, LQL8UQ, LQR4UQ, LQR8UQ, LSDUH, LSDUS, MDY4, ML1Y4, ML3Y3, ML4Y2, T4DLS, UHDLH, UHDUS, UMDX1, UQDLQ, UQDUM, USDS1, 14DLM: All are routines defined in 'GATE".

NCC: Number of division cycle needed. This will be set by 'CVD' and used by 'CVDDIV'

NEGR: = 1, Quotient should be subtracted by 1. This is set by 'CVDDIV' and will be used by 'CVD'.

6.4.1 Convert to Long-fixed point number

1. Name: CVL
2. Functional description:
   This routine is used to convert either a floating point number or a decimal number into a long-fixed point number (full-word). If the converted fixed number is greater than $(2^{31}-1)$ or less then (-2^{31}), an overflow will occur, and the result returned to TP will be the low-order 31 bits with correct sign bit.
3. Formal calling sequence
   CALL CVL
4. Formal Parameter Description: None
5. Implicit Parameter Description:
   Instruction variant (IV), number type (NT) will be used in this routine, the number to be converted is taken from UQ register (exponent in EUM for floating point number). Later on, UQ register will contain the converted number. In case of bogus result, FA contains the error indicator.
6. Subroutines Used:
   DVB, UQDUM, UMDX1, USDS1, ASSIM, SDS, Z4DLM, LMDUQ, UQDLQ, LQR8UQ, LQL8UQ, LQL4UQ, CØMPL
7. Operational Description:
   7.1 English Text:
   There are two types of CVL: floating point to long-fixed point number and decimal to long-fixed point number. For floating point number NT = '10', for decimal number NT = '11', NT is the 8th and 9th bit of the V(INBUS). IBM 360/75 data conversion is also performed for the comparison of results.
   i) Decimal to long-fixed point number
      1) The number to be converted is stored in UQ register bit 9 to bit 64. The sign of that number is in Sign A.
      2) First DVB is used to convert the decimal digits into binary bits and right adjusted in UQ. If UQ_{1-32} is not all zeroes, overflow occurs and ØV indicator is set to 1.

3) If the number to be converted is a negative number, it must be complemented (put in sign-magnitude form) before return to the TP by setting the US register equal to all zeroes and calling the routine 'CØMPL'.

4) In case of overflow, the routine 'SETBØG' is used to set the BØGUS indicator, and to load FA with error indicators.

(ii) Floating Point to Long-Fixed Point Number

1) The number to be converted is stored in UQ register, bits 9 through bit 64. Exponent bits are in EUM; the sign bit in SIGN A.

2) As at point 2, 'II' is the exponent in the integer form. (halfword). (Since EUM is 7 bits long and the first bit is the sign bit, when it is converted to a 16-bit integer, 64 must be subtracted.)

3) For any floating number whose absolute value is less than 1 or greater than 16^{21} (overflow). The converted number will be 0.

4) For any floating point number whose absolute value (x) is $16^{8} < |X| < 16^{21}$, overflow will occur. The converted number is the low-order 31 bits with correct sign bit.

5) The converted fixed point number must be right adjusted in UQ register, if exponent equals 14, no shifting is required, for exponent less than 14, the contents of UQ register must shift right, otherwise shift left. As at point 3, the routine 'FL-NØRL-FX' is called.

6) The maximum fixed point number is $+(2^{31}-1)$ or -2^{31}, point 4 is used to detect the overflow case for $+2^{31}$).

7) For negative number, the result must be the complement, same as in 'convert decimal to fix'.

7.2 Flow Chart

Pages 1-3 of the flowchart for 'CØNV' is for CVL. The symbols used in the flowchart are listed under Section 6.4.

8. Error Condition:

In case of overflow, BØGUS indicator will be set. FA register is used to hold the error indicator.

6.4.2 Convert to Floating Point Number

1. Name: CVF
2. Functional Description:
   This routine is used to convert either a fixed point number (both long and short) or a decimal number into a floating point number.
3. Formal Calling Sequence: CALL CVF
4. Informal parameter description: None
5. Implicit parameter description: The number to be converted is taken from UQ register, sign from SIGNA. The converted number will be stored at UQ register, exponent part in EUL and sign in SR. FA contains the flag bits of the number to be converted and is unchanged.
6. Subroutines used: DVB, UQDLQ, LQL8UQ, LQL4UQ, LQR8UQ, USDS1, UQDUM, UMDX1, Z4DLM, LMDUQ, ASSIM, CØMPL
7. Operational Description:
   7.1 English Text
      i) Convert decimal to floating point number
         1) The number to be converted is in UQ register bit 9 through 64, the sign bit is in SIGNA.
         2) The routine 'DVB' is used to convert the decimal digits in UQ register into binary bits and right adjusted in UQ.
         3) Since the maximum decimal number is 14 digits long when converted to a floating point number, the maximum exponent is 14. The fraction part of a floating point number must be normalized (bit 9-12 not all zeros).
      ii) Convert
         The routine 'NØRM' is then called to normalize the contents of UQ. The sign of the converted number is in SR.
      iii) Fixed point number to floating point number
         1) The number to be converted is taken from UQ register bit 1 through bit 32. In order to share the same normalization logic as in (i), the contents of UQ register is right shifted 8 bits.

2) Then, bit 9 of UQ is the sign bit. For negative number, it must be complemented before normalization. Since bit 9 is the sign bit, bit 9 of US register will be set to 1, and the other bits of US set to 0. The subroutine 'COMPL' is then called.

3) The fixed point number is between -2^{31} and $+(2^{31}-1)$, therefore after conversion the maximum possible exponent is 8. So by setting II=8 and using the routine 'NØRM', the fraction portion of converted number will be normalized in UQ and the correct exponent formed in EUL. The sign will be in SR.

7.2 Flowchart

Page 4-5 of the flowchart for 'CØNV' are CVF.

8. Error Condition: None

6.4.3 Convert to Decimal

1. Name: CVD
2. Functional Description:
   This routine is used to convert either a fixed point number (both short and long) or a floating point number into a decimal number (Illiac III format). In case of overflow, the bogus result will be the low order 14 digits.
3. Formal Calling Sequence: CALL CVD
4. Formal parameter description: None
5. Implicit parameter description: The number to be converted is taken from UQ register (exponent in EUM). The converted number is in UQ register bit 9 through 64, sign bits are in UQ bit 1-8. In case of overflow, FA contains the error indicators, otherwise FA is unchanged. ICC, the no. of division cycles, will be set by this routine and used by 'CVDDIV'; NEGR will be set by 'CVDDIV' and used by this routine.
6. Subroutines Used: CØMPL, UQDLQ, LQR4UQ, LQR8UQ, LQL8UQ, SDS, LHL8UH, LHL4UH, CVDDIV, LMR4M, Z4DLM, LMDUQ
7. Operational Description
   7.1 English Text
   In order to convert a fixed point or floating point number to a decimal, first thing is to left adjust the binary bits in UQ register, then divide it by 10. The portion of fix-division in the 'DIV' routine can be used for this purpose, the entry name is 'CVDDIV'. The only difference is that the M register is not used in CVDDIV and only the remainder will be assimilted, the quotient is in the SDS format.

   Since in fix-division the sign of remainder is the same as the dividend, but in CVDDIV, the remainder is always positive. In order to eliminate the effect of SIGNA (which contains the sign of dividend), the sign is temporarily kept in SR and SIGNA is set to 0, as shown in point 7.

1) For a fixed point number, the number to be converted is in UQ register bit 33 through 64, and bit 33 is the sign bit. For negative number, it must be complemented before conversion. By setting bit 33 of US register to be 1, other bits to be 0, NEGØ to 1 and then calling the routine 'CØMPL', the complemented number will be in UQ bit 33-64.

2) For a floating point number, in order to share the same logic of CVD as for fixed point number, the fraction part of that floating point number will first be right adjusted as fixed point number by using the routine 'FL-NØRL-FX'.

   Since the decimal number can have only 14 digits, if the absolute value of the floating number to be converted is greater than $10^{14}-1$, an overflow will occur and ØV is set to 1.

3) From here on, the following steps are common for both fixed point or floating point conversion. ND is a counter for the number of decimal digits which have been formed. If ND > 14, the conversion process will terminate.

4) The contents of UQ register will be used as dividend and divided by 10 using 'CVDDIV'. The decimal digits are generated from the least significant to the most significant.

5) The remainder after each division is a decimal digit. The M register is used to store the converted decimal digit. Every time a division is completed, the contents of M register will be right shifted 4 bits, the new remainder (bit 9-12 of LM register) will be inserted into bit 9-12 of M, as shown at point 8.

6) The quotient formed in UH and UQ register will be used as new dividend. The steps (3) - (6) will be repeated until either ND > 14 or the UQ register is zero.

7) The SR, which contains the sign of the number to be converted, is used to set the sign of result. For positive numbers, bit 5-8 of UQ is set to 1010, for negative numbers it is set to 1011.

8) Steps (8) - (10) will describe the detail of division. Since for the Model Division used in the Illiac III, the dividend is aligned between 1/2 and 1. In order to use the model division, the contents of UQ, UH register must be left shifted accordingly, as shown at point 9.

9) NCC is number of division cycle needed. This parameter will be used by the routine 'CVDDIV'. To save execution time, no parameter will be bended, so NCC must be declared as an external variable in routines 'CVD', 'DIV' and 'AUSIM'.

10) After the division is completed, the indicator NEGR is set by 'CVDDIV'. If NEGR = 1, the quotient (in UH and UQ) must be subtracted by 1, as shown at point 10, but not assimilated.

7.2 Flowchart

Pages 6-9 of the flowchart of 'CØNVS' are for CVD.

8. Error Condition:

In case of overflow, the bogus result is the low order 14 digits with correct sign bits. The ØV, BØGUS will be set, FA contains the error indicators.

9. Local Procedures:

LMR4M: The contents of LM register are right shifted 4 bits and transmitted into M register.

6.4.4 Procedure Routines Defined within 'CØNVS'

i)

1. Name: DVB
2. Functional Description: This routine is used to convert the decimal digits, stored in bits 9-64 of UQ register, into binary bits, which are right adjusted in UQ. This routine will be used by CVL or CVF whenever the number type is decimal.
3. Formal Calling Sequence: CALL DVB
4. Formal Parameter Description: None
5. Implicit Parameter Description: None
6. Subroutines Used: UQDLQ, LQL8UQ, LQL4UQ, USDS1, UMDX1, SDS, ML3Y3, ML1Y4, LMDUQ, LSDUS, Z4DLM, LMDM, ASSIM.
7. Operational Description:

7.1 English Text

1) The decimal digits to be converted are kept in UQ register, bits 9-64.
2) At point 11, for all zero decimal digits the converted binary bits are also zero.
3) For decimal digits other than zero, the maximum number of digits to be converted is 14. ICC is a counter.
4) At point 12, UQ is left shifted to eliminate leading zeros, ICC is decreased accordingly.
5) Now, the decimal number is in the form: $X_0, X_1 \ldots X_{ICC}$ starting from bit 5 of UQ; where X stands for one decimal digit, X_0 is most significant digit, and X_{ICC} is the least significant digit.

This routine is to evaluate the following equation: $((((X_0).10+X_1).10)+X_2).\ 10+\ldots).10+X_{ICC}$.

6) The most significant digit is taken from bits 5-8 of UQ register. M register is used to store the intermediate result. Initially M register contains X_0. Bit 9-12 of the UQ register will be transmitted to bits 61-64 of UM register. Then the intermediate result (M register) is multiplied by 10 and added to the contents of UM register (by left shift M register by 3, then by 1). The new intermediate result is

kept in M register again, as shown from point 13 to 14.

7) If more decimal digits remain to be processed, the contents of UQ register are then left shifted 4 bits. Step (6) is repeated until all decimal digits have been processed.

8) The converted binary bits are right adjusted in the UQ register.

7.2 Flowchart

Pages 10-11 of the flowchart 'CØNVS' are for DVB.

8. Error Condition: None

(ii)

1. Name: CØMPL
2. Functional Description: This routine is used to complement the contents of UQ register. This routine can be called by CVL, CVF and CVD.
3. Formal Calling Sequence: CALL CØMPL
4. Formal Parameter Description: None
5. Implicit Parameter Description: US and NGGØ must be set by the calling program before this routine is called.
6. Subroutines used: USDS1, UQDUM, UMDX1, ASSIM, Z4DLM, LMDUQ
7. Operational Description:

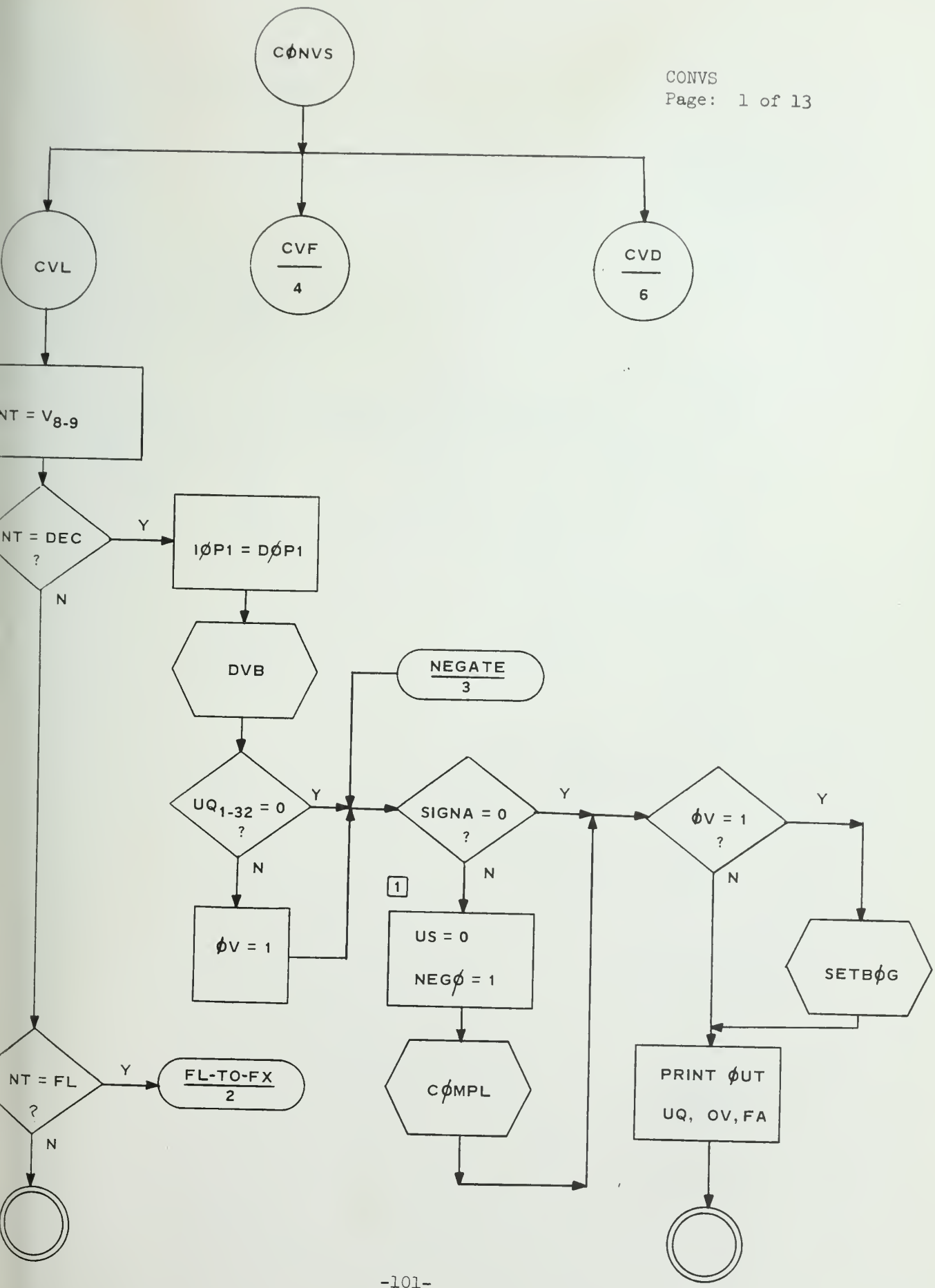
7.1 English Text:

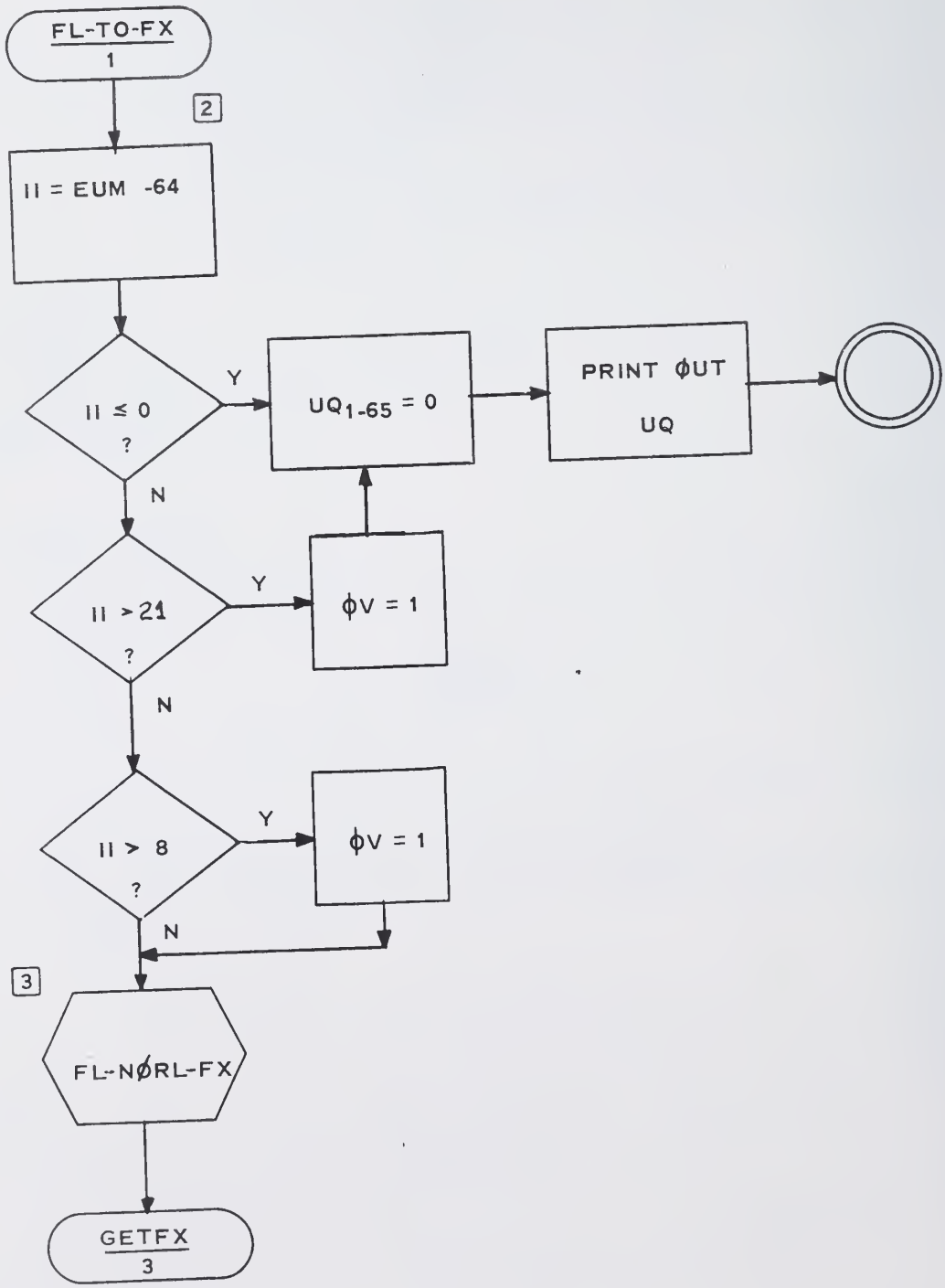
US and NEGØ are set by the calling program, the exclusive OR of these two is used as the S-input to Signed-digit subtractor (SDS). UQ is used as the X-input to SDS: Y and NEG1 are set to 0, then the assimination routine 'ASSIM' is called. Finally the complimented result is put back into UQ register.

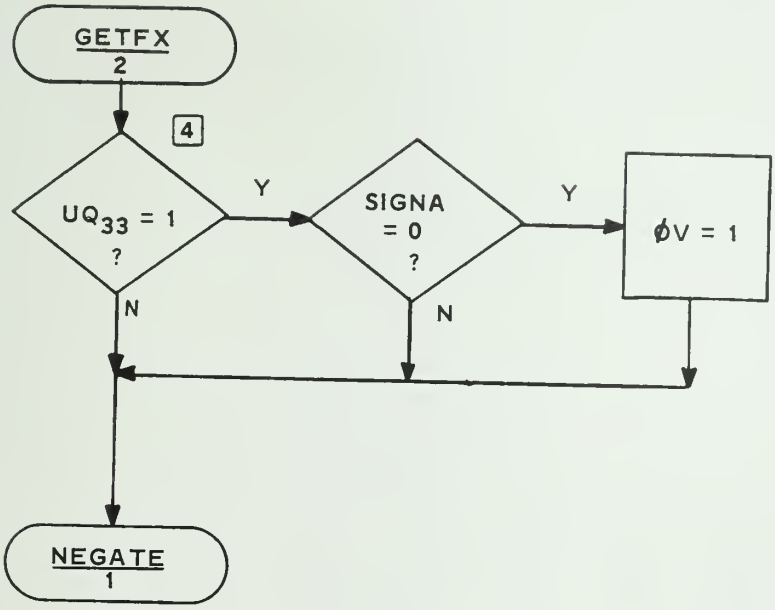
7.2 Flowchart

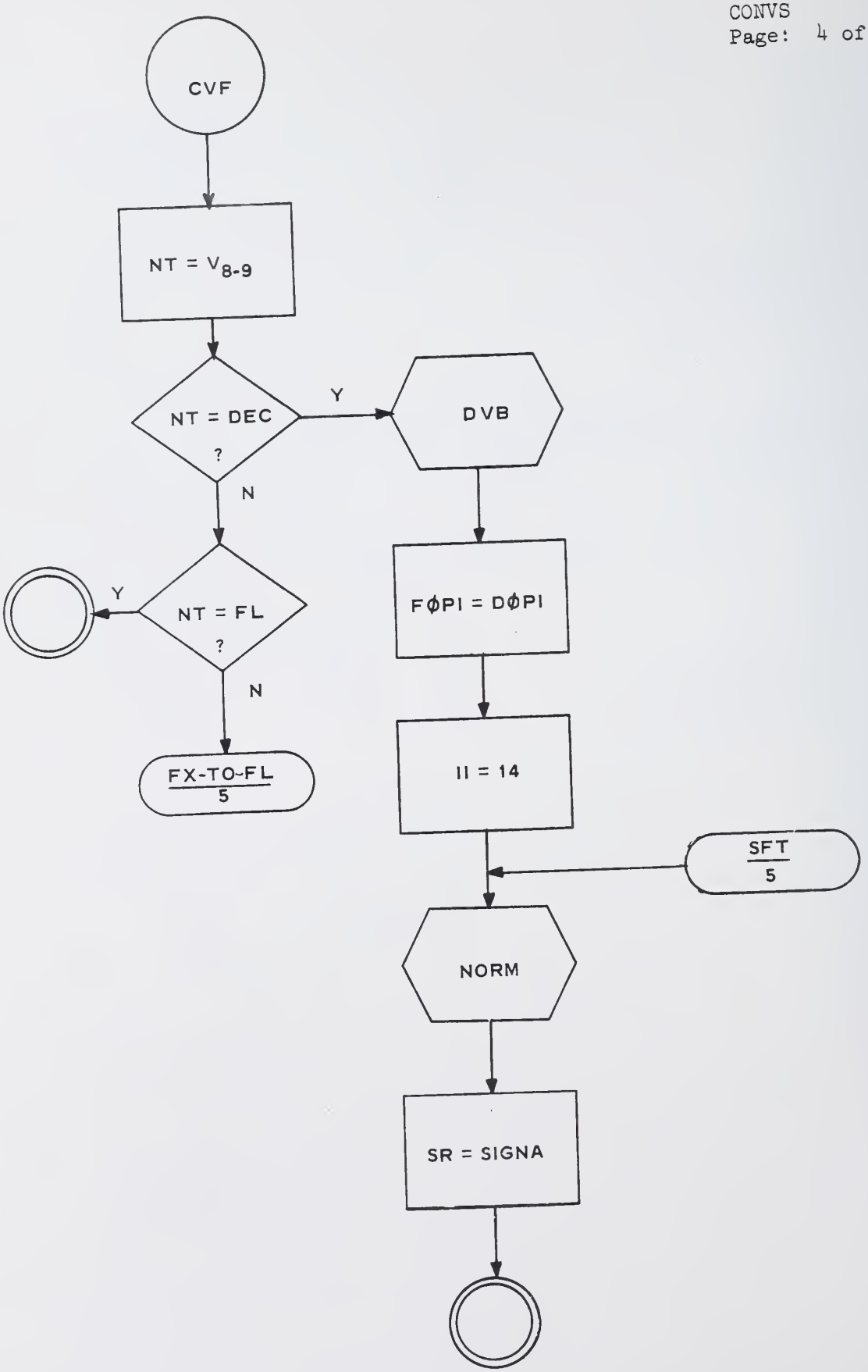
Page 12 of the flowchart 'CØNVS' is for CØMPL.

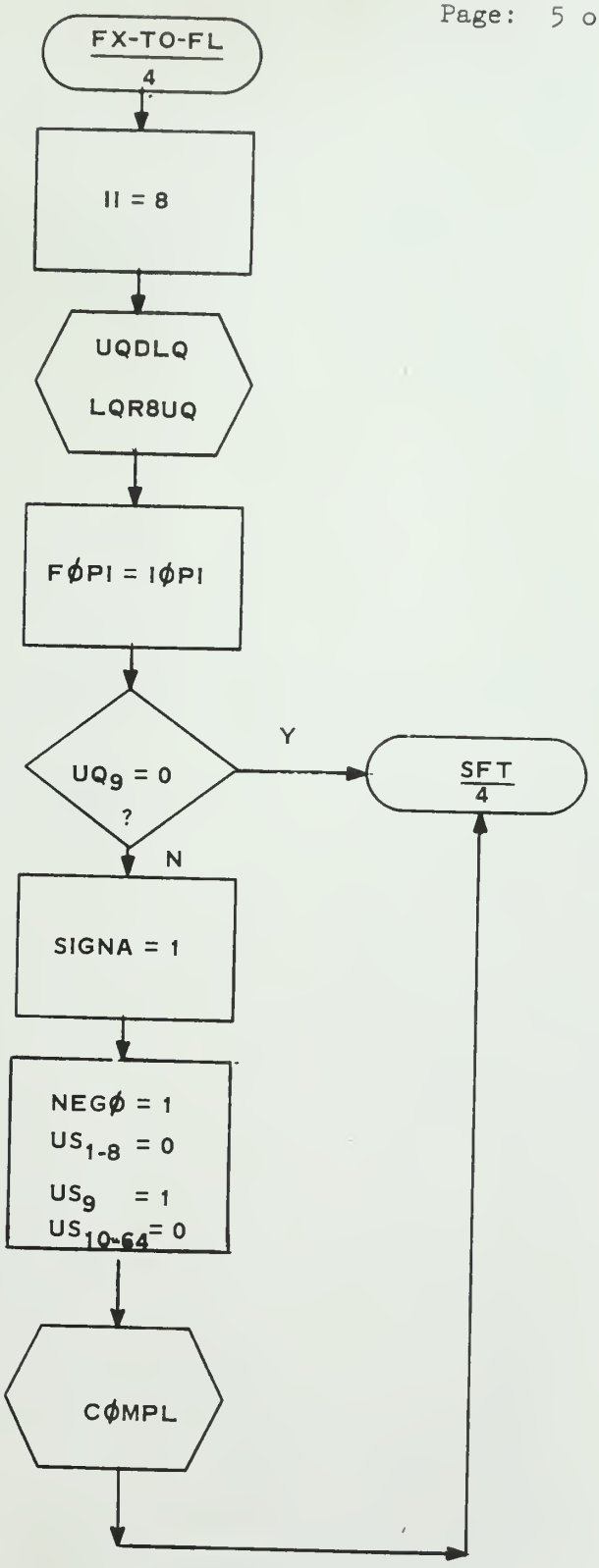
8. Error Condition: None

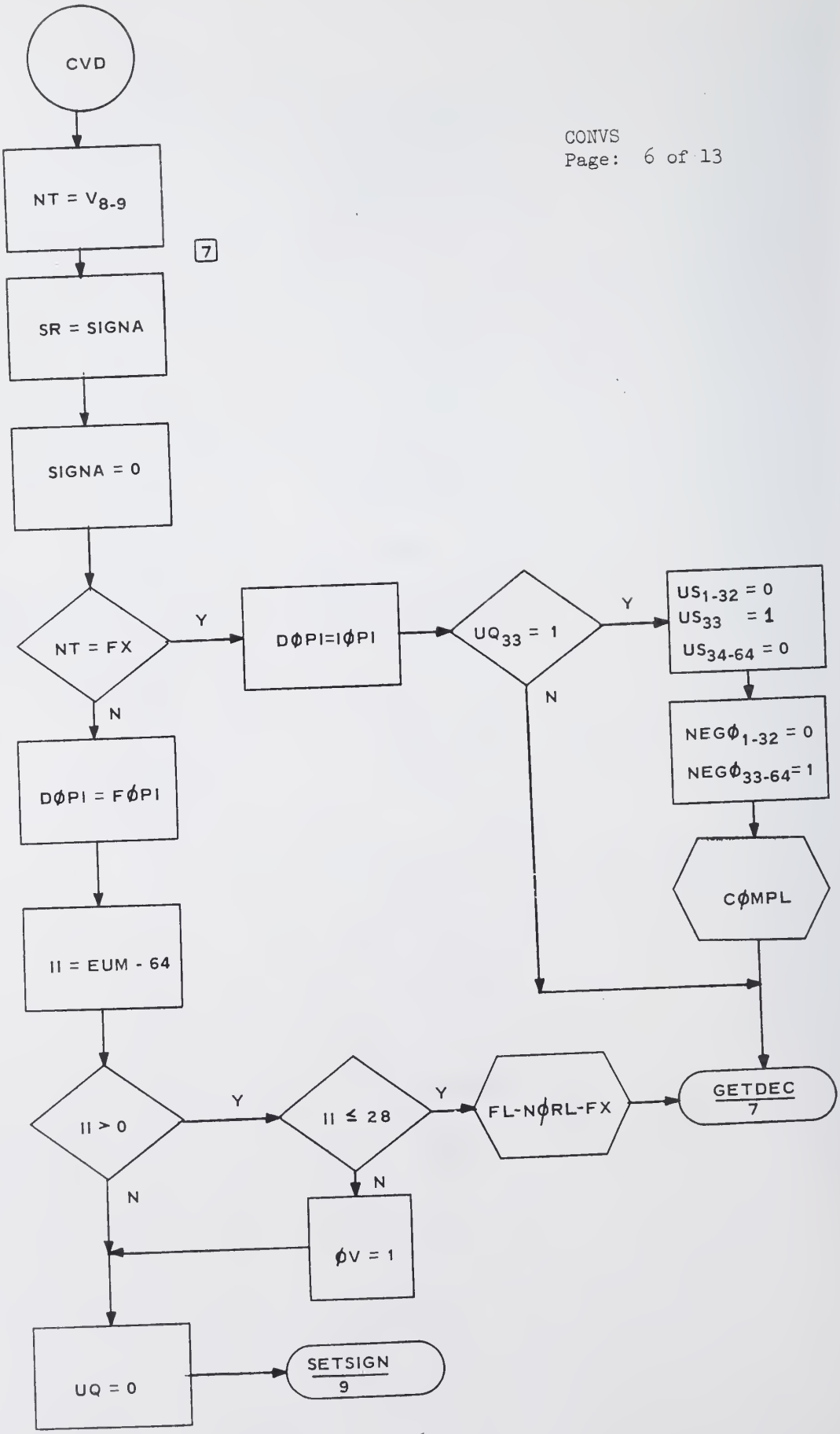

CØNVS
CVL
CVF
4
CVD
6
NT = V8-9
NT = DEC
?
Y
N
IØP1 = DØP1
DVB
NEGATE
3
UQ1-32 = 0
?
Y
N
SIGNA = 0
?
Y
N
ØV = 1
?
Y
N
1
ØV = 1
US = 0
NEGØ = 1
CØMPL
SETBØG
PRINT ØUT
UQ, OV, FA
NT = FL
?
Y
N
FL-TO-FX
2

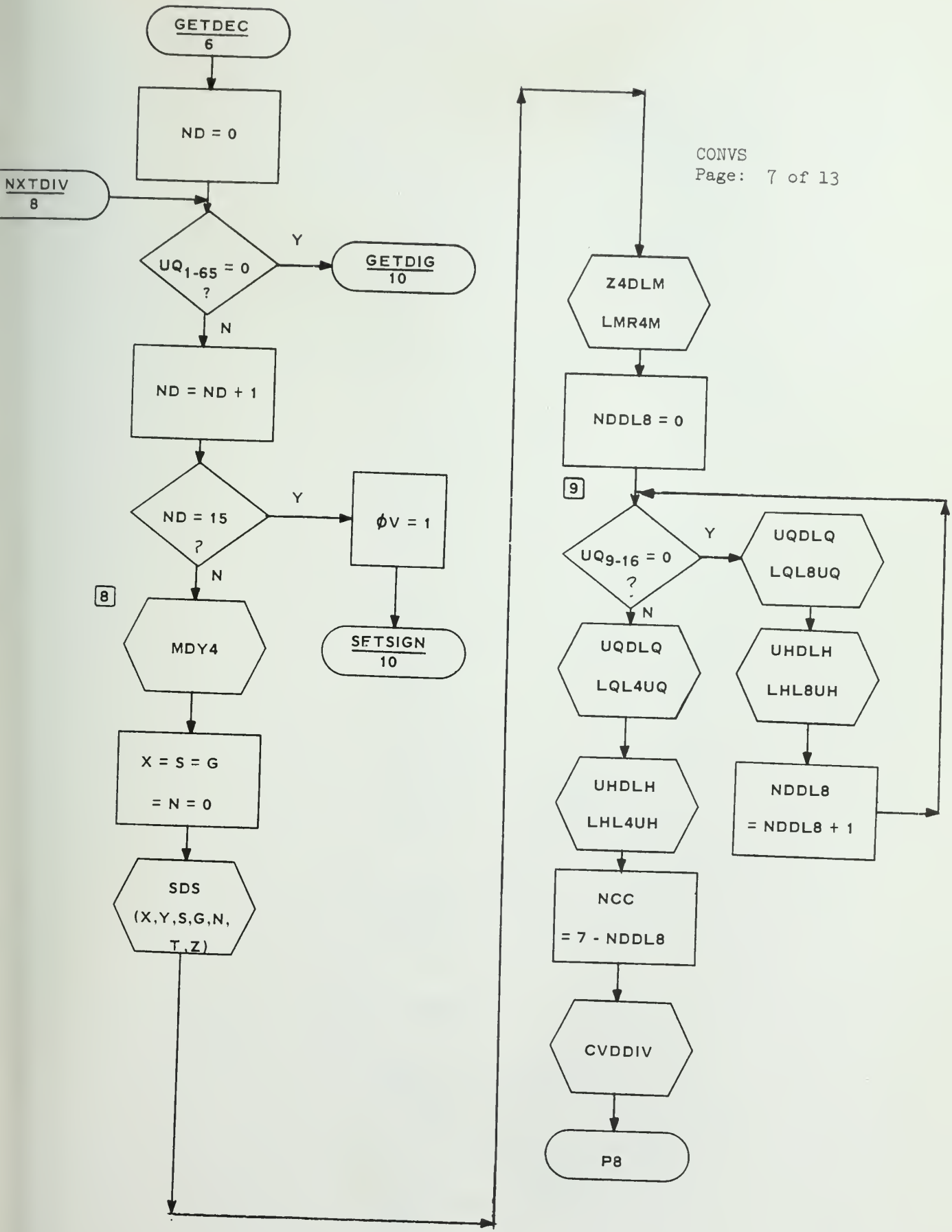

FL-TO-FX
1
2
II = EUM -64
II ≤ 0
?
Y
UQ1-65 = 0
PRINT ϕUT
UQ
N
II >21
?
Y
ϕV = 1
N
II > 8
?
Y
ϕV = 1
N
3
FL-NØRL-FX
GETFX
3

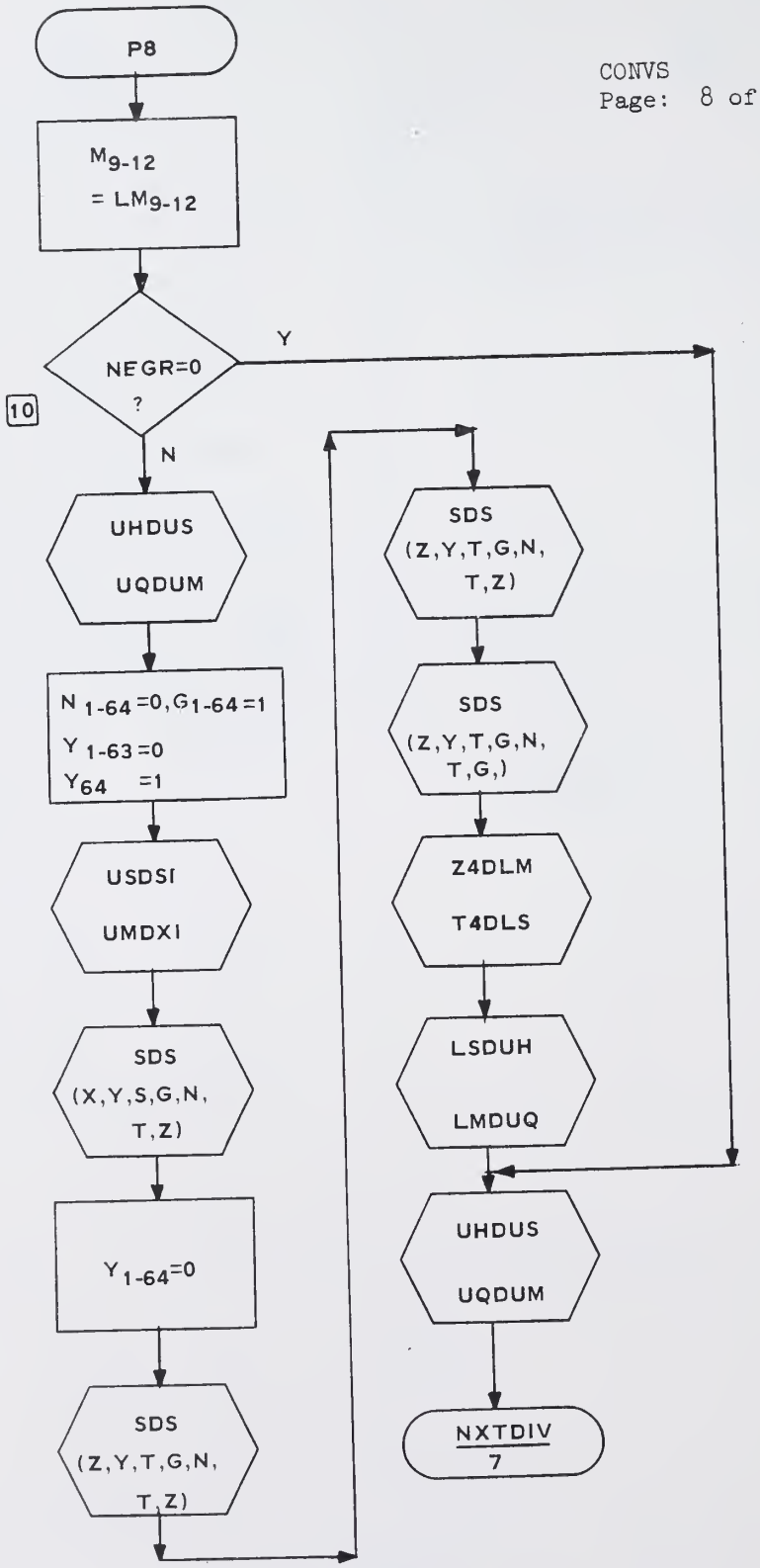

GETFX
2
4
UQ33 = 1
?
Y
SIGNA
= 0
?
Y
ϕV = 1
N
N
NEGATE
1

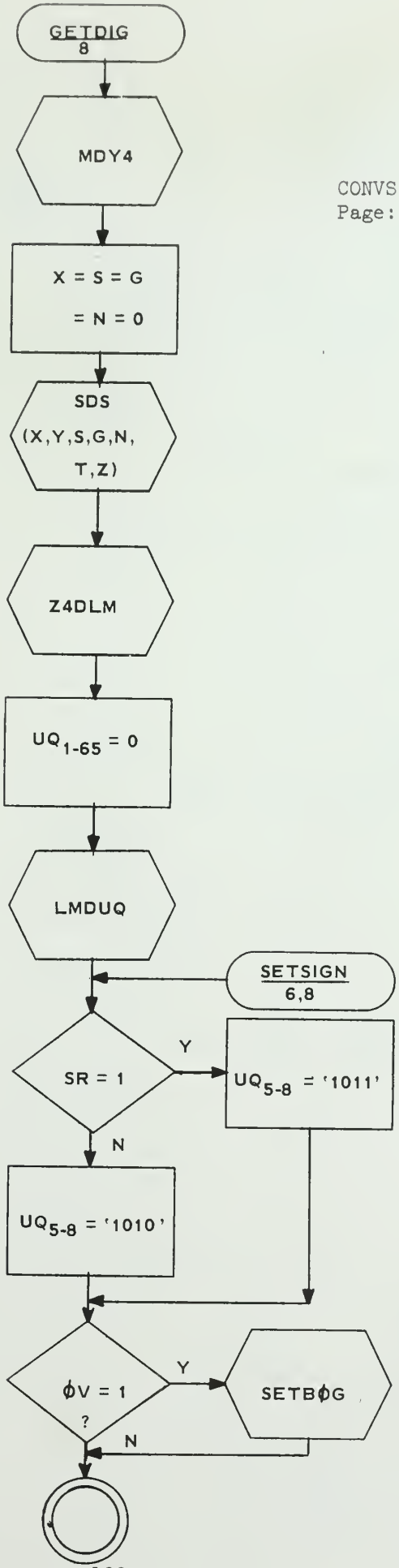

CVF
NT = V8-9
NT = DEC ?
Y
DVB
N
NT = FL ?
Y
N
FX-TO-FL 5
FØPI = DØPI
II = 14
SFT 5
NORM
SR = SIGNA

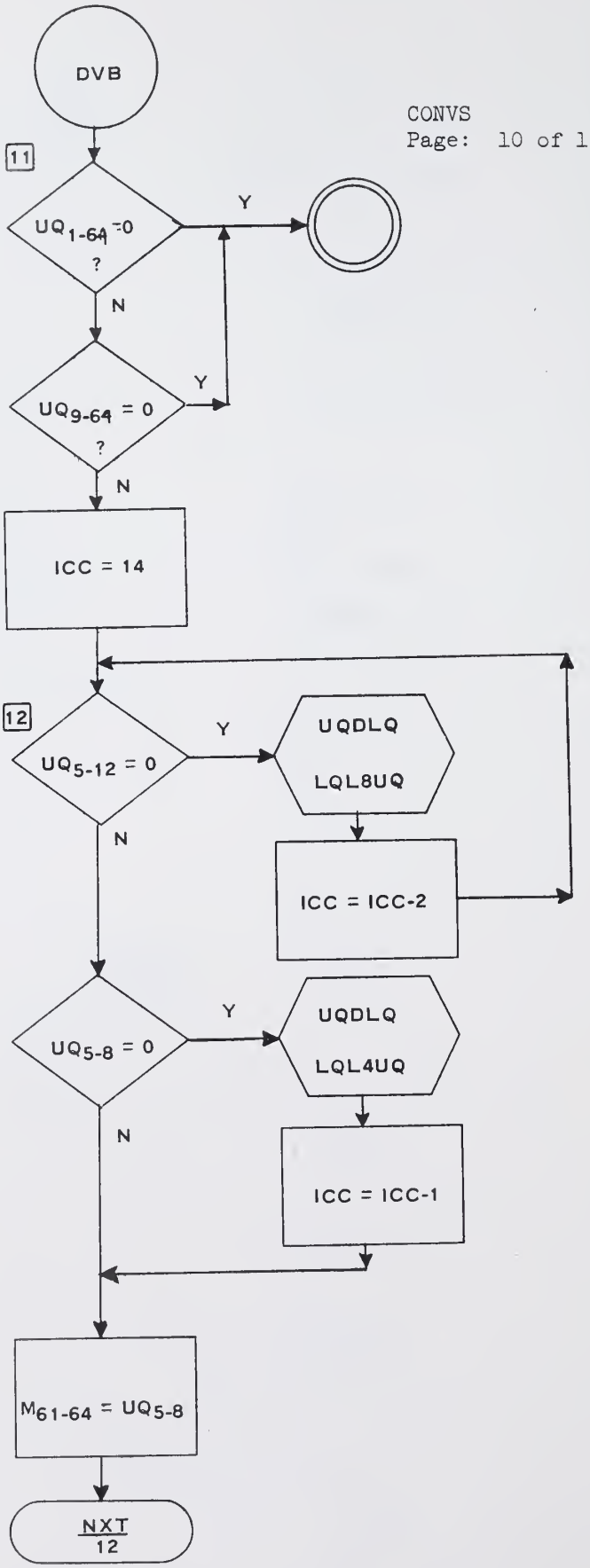

FX-TO-FL
4
II = 8
UQDLQ
LQR8UQ
FØPI = IØPI
UQ9 = 0
?
Y
SFT
4
N
SIGNA = 1
NEGØ = 1
US1-8 = 0
US9 = 1
US10-64 = 0
CØMPL

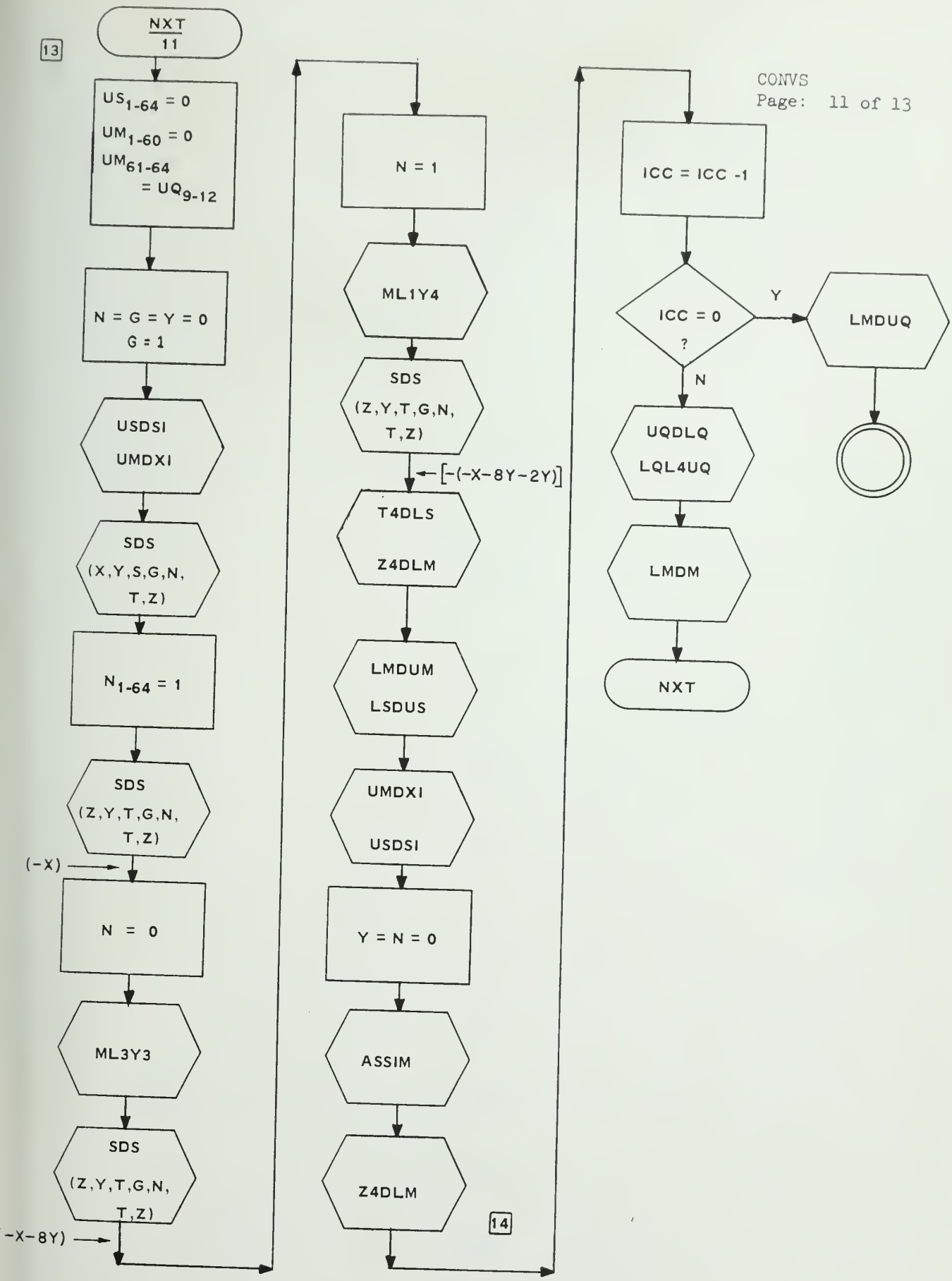

CVD
NT = V8-9
7
SR = SIGNA
SIGNA = 0
NT = FX
Y
N
DØPI=IØPI
UQ33 = 1
Y
N
US1-32 = 0
US33 = 1
US34-64 = 0
NEGØ1-32 = 0
NEGØ33-64 = 1
CØMPL
DØPI = FØPI
II = EUM - 64
II > 0
Y
N
II ≤ 28
Y
N
FL-NØRL-FX
GETDEC
7
ØV = 1
UQ = 0
SETSIGN
9

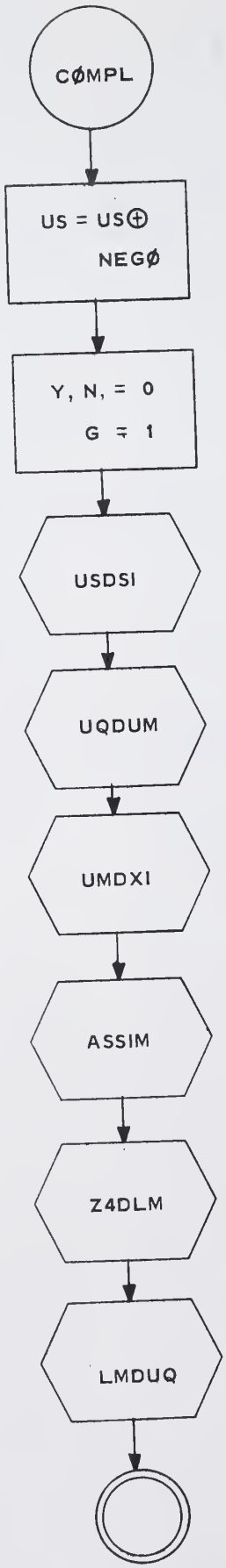

GETDEC
6
ND = 0
NXTDIV
8
UQ1-65 = 0
?
Y
GETDIG
10
N
ND = ND + 1
ND = 15
?
Y
ϕV = 1
SFTSIGN
10
N
8
MDY4
X = S = G
= N = 0
SDS
(X,Y,S,G,N,
T,Z)
Z4DLM
LMR4M
NDDL8 = 0
9
UQ9-16 = 0
?
Y
UQDLQ
LQL8UQ
UHDLH
LHL8UH
NDDL8
= NDDL8 + 1
N
UQDLQ
LQL4UQ
UHDLH
LHL4UH
NCC
= 7 - NDDL8
CVDDIV
P8

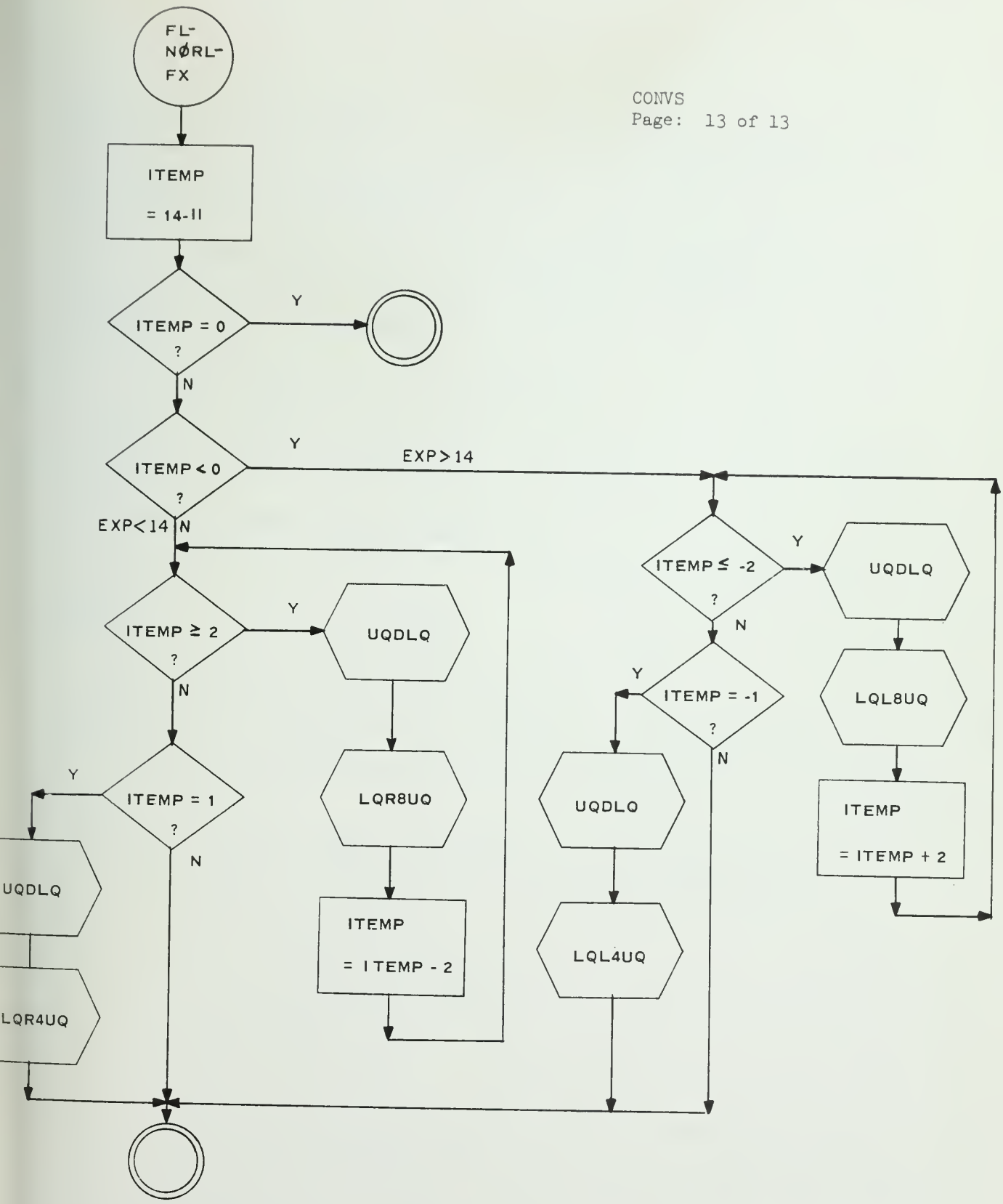

P8
M_{9-12} = LM_{9-12}
NEGR=0 ?
Y
N
10
UHDUS
UQDUM
N_{1-64}=0, G_{1-64}=1
Y_{1-63}=0
Y_{64} =1
USDSI
UMDXI
SDS (X,Y,S,G,N, T,Z)
Y_{1-64}=0
SDS (Z,Y,T,G,N, T,Z)
SDS (Z,Y,T,G,N, T,Z)
SDS (Z,Y,T,G,N, T,G,)
Z4DLM
T4DLS
LSDUH
LMDUQ
UHDUS
UQDUM
NXTDIV 7


GETDIG 8
MDY4
X = S = G = N = 0
SDS (X,Y,S,G,N, T,Z)
Z4DLM
UQ1-65 = 0
LMDUQ
SETSIGN 6,8
SR = 1
Y
UQ5-8 = '1011'
N
UQ5-8 = '1010'
ϕV = 1 ?
Y
SETBϕG
N


DVB
11
UQ1-64 =0 ?
Y
N
UQ9-64 = 0 ?
Y
N
ICC = 14
12
UQ5-12 = 0
Y
UQDLQ
LQL8UQ
N
ICC = ICC-2
UQ5-8 = 0
Y
UQDLQ
LQL4UQ
N
ICC = ICC-1
M61-64 = UQ5-8
NXT
12


13
NXT
11
US1-64 = 0
UM1-60 = 0
UM61-64
= UQ9-12
N = G = Y = 0
G = 1
USDSI
UMDXI
SDS
(X,Y,S,G,N,
T,Z)
N1-64 = 1
SDS
(Z,Y,T,G,N,
T,Z)
(-X)
N = 0
ML3Y3
SDS
(Z,Y,T,G,N,
T,Z)
(-X-8Y)
N = 1
ML1Y4
SDS
(Z,Y,T,G,N,
T,Z)
[-(-X-8Y-2Y)]
T4DLS
Z4DLM
LMDUM
LSDUS
UMDXI
USDSI
Y = N = 0
ASSIM
Z4DLM
14
ICC = ICC -1
ICC = 0
?
Y
LMDUQ
N
UQDLQ
LQL4UQ
LMDM
NXT


CØMPL
US = US⊕
NEGØ
Y, N, = 0
G = 1
USDSI
UQDUM
UMDXI
ASSIM
Z4DLM
LMDUQ


FL-
NØRL-
FX
ITEMP
= 14-II
ITEMP = 0
?
Y
N
ITEMP < 0
?
Y
EXP>14
EXP<14
N
ITEMP ≥ 2
?
Y
UQDLQ
LQR8UQ
ITEMP
= ITEMP - 2
N
ITEMP = 1
?
Y
UQDLQ
LQR4UQ
N
ITEMP ≤ -2
?
Y
UQDLQ
LQL8UQ
ITEMP
= ITEMP + 2
N
ITEMP = -1
?
Y
UQDLQ
LQL4UQ
N

(iii)

1. Name: FL - NØRL- FX
2. Functional Description: Whenever a floating point number is to be converted to a double word fixed point number (right adjust), this routine is used to right shift or left shift the contents of UQ register according to the exponent of the floating point number.
3. Formal Calling Sequence: CALL FL-NØRL-FX
4. Formal Parameter Description: None
5. Implicit Parameter Description: II, the exponent of the floating point number to be converted, is used as the key for shifting in this routine. The exponent is extended into 16 bits and stored in II.
6. Subroutines used: UQDLQ, LQR4UQ, LQR8UQ, RQL4UQ, LQL8UQ
7. Operational Description:

   7.1 English Text

   The function portion is stored in bits 9-64 of UQ register, therefore when the exponent equals 14, no shifting is required. If exponent is less than 14, UQ register must right shift (14-EXP)*4 bits. If exponent is greater than 14, UQ register must be left shifted (EXP-14)*4 bits.

   7.2 Flowchart

   Page 13 of the flowchart of 'CONVS' is for this routine.

8. Error Condition: None

VII. How to Use the Simulator

7.1 Execution of the Simulation

The execution starts at the main program 'AUSIM'. The flow of execution is determined by the input data deck, so the set up of the data deck will be described first.

(a) The first item to be read in is

PRINT = X ;

X may be either 0 or 1, 0 means to skip the print out of SDS routine. It is data-directed input and thus may start at any column n and must be terminated by a semicolumn ';'.

(b) Seven items must be read in. They are list-directed and separated by comma ',', or blank(s):

Item 1 is IV : 'XXXX' B

Item 2 is NT: 'XX' B

Item 3 is an indicator IND X

X = 0 means items 4 and 6 are in conventional format.

X = 1 means items 4 and 6 are in the internal bit representation of Illiac III.

Item 4 is the first operand

IND = 0 Conventional fixed, floating or decimal no.

IND = 0 'XXXXXX' B

64 bits for FL, DEC

32 bits for long FX

16 bits for short FX

Item 5 is the flag bits of item 4 'XXXXXX' B

8 or 4 or 2 bit

Item 6 is the second operand. It should be in the same form as item 4.

Item 7 is the flag bits of item 6. It should be in the same form as item 5.

(c) Repeat (b) as many as needed.

7.2 Modification of the Programs

Almost every routine in this simulator is concerned with bit manipulation, therefore when any program needs to be changed, the data type and length of data involved in both sides of the assignment statement must be considered. The PL/1 built in function SUBSTR, when used in the left hand side of an equation, sometimes didn't work as expected at present time. Some additional statements in this simulator (for example, temporary storage) is just to avoid this trouble. When changes involve SUBSTR, be very careful.

To save execution time, with all routines except SDS, there is no parameter binding at execution time when the program is called. All variables (registers, counters, indicators) needed for communication between sub-programs are declared as EXTERNAL in both the sub-programs involved as well as in the main program AUSIM. The main program has to have all the EXTERNAL variables, declared in the subprogram directly or indirectly called by it, also declared in it.

In this simulation, all the logic which may be shared by more than one arithmetic order are defined as subroutines. To save execution time, no parameter is bound when the routine is called, but rather some parameter defined as external variables, are set before calling the sub-routine. When any modification is made, be sure all the implicit requirements of subroutines are met.

At present time, to transmit the results from the AU back to TP via ØUTBUS has not been simulated yet. It is suggested to add this in the main program (AUSIM) rather than to change every arithmetic order program. That is, to add it right after the 'CALL' statement for each arithmetic routine and before going back to the beginning of the AUSIM (clear all registers).

7.3 Set up of program decks

Since POLY (with the name POLY X used in this simulation) has not been simulated yet, a dummy procedure is involved in each execution.

(a) All other programs are in object decks

```
ID CARD
//        EXEC      PL1 (optional PARM)
//PL1.SYSIN    DD   *
PØLYX : procedure;                    dummy procedure for POLY
        return; end;
/*
//      EXEC     LKGØPL1
//LKED.SYSIN     DD    *
object decks for all prog.
(AUSIM, GATE, SDS, PRØP
ADDX, MPY, DIV, CØNVS)
/*
//GØ.SYSIND      DD     *
Data Deck
/*
```

If some subprograms need to be modified and debugged, the set up may be changed to:

```
ID card
//     EXEC    PL1
// PL1.SYSIN       DD     *
                                Repeat as many source program needed
source deck of one program
/ *
// LKGD.SYSIN      DD     *
object decks of other prog.
/*
//GØ.SYSIN      DD     *
Data Deck
/ *
```

The only requirement is that all the external procedures of the AU simulation (as listed in the general description) must be involved. They may be either in source deck (or dummy procedure) or in object deck.

(b) All object decks are on disk

```
ID card
//     EXEC   PL1LKGØ (optional PARM)
//PL1.SYSIN      DD    *
Dummy procedure for POLY
/*
//LKED.SYSLIB    DD   DSNAME=SYS1.PL1LIB, DISP = ØLD
//              DD   DSNAME = user. P1142.ILLIAC3.AU,
//              DISP = OLD, VØLUME = USER = UIDCS3
// LKED.SYSIN    DD    *
b INCLUDE SYSLIB (AUSIM, GATE, SDS, PRØP, ADOX, MPY, DIV, CØNVS)
b ENTRY IHENTRY
/*
//GØ.SYSIN    DD    *
Data Deck
/ *
```

(c) To ADD new object decks on disk or replace some old objects on disk by some new object Deck.

When a routine has been changed, it may use together with the object deck of other routines to debug as in (a) after it has been debugged put on disk; or it may be compiled first and replace the old object on disk and use (b) to debug.

```
ID CARD
//        EXEC      LKGDPL1                                          col. 72
//LKED.SYSLMØD   DD    DSNAME = USER.P1142.ILLIAC3,AU                 X
//               UNIT = DISK, VØLUME = SCR = UIDCS3,                  X
//               SPACE = (TRK, (40, 10, 10)), DISP = (ØLD,KEEP)
// LKED.SYSIN    DD   *

object deck of one program

b NAME bb  XXXXXXX (R)
           Name of that program          Repeat for every program
/*
```

NOTE: As of September 1968 the AUSIM program decks were stored in the card files in Room 200 (Dr. Lansford's office).

```
//              EXEC    PL1
//PL1.SYSIN     DD  *
  AUSIM :  PROCEDURE  OPTIONS(MAIN)  ;
   /* THIS PART DEFINES THE STURCTURE OF REGISTERS*/                    */
    /*    SYMBOL  MAP:                                                  */
    /* ***  REGISTERS       LENGTH      NAME                            */
    /*       M               56 BITS     M ,M_S                        */
    /*      US               64          US, US-S                      */
    /*       UM              64          UM, UM_S                      */
    /*       LS              64          LS, LS_S                      */
    /*       LM              64          LM, LM_S                      */
    /*       UH              64          UH, UH_S                      */
    /*       UQ              64          UQ, UQ_S                     1*/
    /*       LH              56          LH, LH_S                      */
    /*       LQ              56          LQ, LQ_S                      */
    /*       V               32          V , V_S                       */
    /*       F                8          F , F__S                      */
    /*       EUU              8          EUU,EUU_S     EXP UNIT        */
    /*       EUM              8          EUM,EUM_S     EXP UNIT        */
    /*       FA               8          FA,FA_S       FLAG            */
    /*       FB               8          FB,FB_S       FLAG            */
        /*          STRUCTURE FORM OF ALL REGISTERS                    */
        /*       1H   , I BYTE(7) ,          3  BBIT(8)      56 BITS    */
        /*       1UH , 2 BYTE(8) ,  3 BBIT(8)                64 BITS    */
        /*       1US       LIKE   UH                         64 BITS    */
        /*       1UM       LIKE   UH                         64 BITS    */
        /*       1LS       LIKE   UH                         64 BITS    */
        /*       1LM       LIKE   UH                         64 BITS    */
        /*       1LH       LIKE   M                          56 BITS    */
        /*       1UQ       LIKE   UH                         64 BITS    */
        /*       1LQ       LIKE   M                          56 BITS    */
        /*       1EUM    , 2  BBIT(7)                         7 BITS    */
        /*       1EUU      LIKE   EUM                         7 BITS    */
        /*       1FA      ,2  BBIT (8)                        8 BITS    */
        /*       1FB       LIKE   FA                          8 BITS    */
        /* II= TEMPORARY REGISTER TO HOLD SUM OR DIFF. OF EXPONENT    */
       DCL M BIT(64) PACKED EXT,  (LQ,UQ) BIT(65) PACKED EXT,  II EXT,
           (UH,LH,US,LS,UM,LM,X,Y,S,T,Z,N,G)  BIT(64) PACKED EXT,
         (P,B) BIT(64) PACKED EXT,
           V BIT(50) EXTERNAL,
           (EUU,EUM) BIT(7) EXTERNAL,  EUL  BIT(7) EXTERNAL,
           (FA,FB)  BIT(8) EXTERNAL,
           (SIGNA,SIGNB) BIT(1) EXTERNAL, POLIP BIT(1) EXTERNAL,
           (OV,UN,LSG,ID,GT,EQ,LT,FM,BOGUS) BIT(1) EXT,
           (SDB,SR) BIT(1) EXTERNAL,
           AEXPGT  FIXED BIN EXTERNAL;
        DCL NEGR EXT,  NCC EXT;
        DCL    IV(4) BIT(1) ,  IV_S  BIT(4) ,
               NT(2) BIT(1) ,  NT_S  BIT(2)  ,
               INDIVR  BIT(4)  ,   NTDNTR  BIT (2)  ,
               TEMP(4)  BIT (32) ,  (T1,T2)  BIT(64) ;
        DCL    (WRDCT,NOP,CT) FIXED  BIN(15);
        DCL    IOP(2) FIXED BIN(31) EXTERNAL,
               FOP(2)  FLOAT BIN(53)  EXTERNAL,
               DOP(2)  DECIMAL FIXED (15) EXTERNAL,
               OPF(2)  BIT(8)   , FLAG(4) BIT(4) ;
       DCL  (TEST1,TEST2)  BIT(64);
```

```
    DCL   C(2) BIT(64);
     DCL  PRINT FIXED BIN EXT;
/*    END OF REGISTER DEFINED*/
/*    THIS PART IS A TEST FOR ALL ROUTINES     */
 /*  READ IN A FLOATING NO. AT  INS_ OR INS_S , MOVE TO V_S  */
     F3 :  FORMAT  (SKIP,A, X(4),B(64));
     F2 : FORMAT (B(32),X(1));
     F1 :  FORMAT  (SKIP, A, X(4), B(64));
     /*  CLEAR ALL REGISTERS  &  COUNT   */
     GET DATA (PRINT);
     CLEAR : POLIP='0'B;  /* POLIP WILL BE CLEARED FOR ALL ORDER*/
    KLEAR :  CALL CLRREG; /* EXCEPT FOR POLY                    */
     CLRREG : PROCEDURE   ;      /*  CLEAR  ALL REBS */
              V=(50)'0'B;
    M=(64)'0'B;  UQ,LQ=(65)'0'B ;
     LH,UH,LS,US,LM,UM,X,Y,S,T,Z,N,G=(64)'0'B;
              EUU=(7)'0'B;  EUM=EUU;  SIGNA='0'B;  SIGNB=SIGNA;
              FA=(8)'0'B;  FB=FA;   EUL=EUU;
              WRDCT=0;  CT=1;  NOP=0;
        SDB='0'B ;               SR=SDB;
        AEXPGT=0;
     OV,UN,LSG,ID,GT,EQ,LT,FM,BOGUS='0'B;
              RETURN;   END CLRREG ;
        /* THIS  PART  IS  TO READ  IN IV & NT , DETERMINE  NOP
              THEN  READ  IN  OPRANDS  PUT IN PROPER BUFFER
                              (FIXED , FLOAT  & DECIMAL  )
              THEN  CONVERT TO BIT FORM                          */
TPSIM : GET FILE(SYSIN) LIST (IV_S,NT_S,IND);
     /*  FIRST CARD IV(4BIT),NT( 2 BIT), IND=1 FOR READING IN BIT
         FORM FOR OPRANDS   */
    PUT FILE(SYSPRINT) EDIT ('START SIMULATION') (PAGE,X(40),A);
     CALL TIMER;
          PUT FILE (SYSPRINT) LIST ('IV   ',IV_S,'NT    ', NT_S) SKIP;
     DO I = 1  TO 4  ;
        IV(I)= SUBSTR(IV_S, I,1) ;
        END ;
    T1,T2=(64)'0'B;
     DO I=1 TO 2 ;
     NT(I) = SUBSTR (NT_S,I,1);
     END ;
     IF ¬IV(1)  THEN   NOP=1  ;
                ELSE   NOP=2  ;
/* TEST FOR SUBSEQUENT POLY, ONLY ONE OPRAND IS NEEDED */
     IF POLIP THEN NOP=1;
        /*  START  READING   OPRANDS   */
     IF ¬NT(1)  THEN   DO  ;
    IF   IND=1 THEN DO ;
        IBIT : GET FILE(SYSIN) LIST ((C(I),OPF(I) DO I=1 TO NOP));
        UNSPEC(IOP(1))=C(1);   T1=C(1);
        UNSPEC(IOP(2))=C(2);   T2=C(2);
        PUT FILE (SYSPRINT) EDIT ('OP1= ',IOP(1),C(1),
         'OP2= ',IOP(2),C(2)) (SKIP,A,F(16,0),SKIP,X(7),B(64));
        GO TO  CONV;  END;
         FX : GET FILE(SYSIN)   LIST (( IOP(I),OPF(I)  DO I = 1 TO
                                      NOP )) ;
     PUT FILE(SYSPRINT ) LIST ('FIXED   ' ,( IOP(I),OPF(I)  DO I=
           1 TO NOP )) SKIP;
```

```
        T2= UNSPEC(IOP(2))  ;
        T1= UNSPEC(IOP(1))  ;
        /* FX NO. MUST BE RIGHT ADJUST IN THE LEFT WORD */
     GO TO  CONV ;
   END  ;
 IF  NT(1)  & ¬NT(2)  THEN DO ;
IF IND=1 THEN DO ;
    FBIT : GET FILE(SYSIN) LIST ((C(I),OPF(I) DO I=1 TO NOP));
    UNSPEC( FOP(1))=C(1);   T1=C(1);
    UNSPEC( FOP(2))=C(2);   T2=C(2);
    PUT FILE(SYSPRINT) EDIT ('OP1= ',FOP(1),C(1),'OP2= ',
      FOP(2),C(2)) (SKIP,A,E(16,6),SKIP,X(7), B(64));
    GO TO CONV;  END;
     FL : GET FILE(SYSIN)   LIST (( FOP(I),OPF(I)  DO I= 1 TO
                                  NOP )) ;
    PUT FILE (SYSPRINT) LIST ('FLOAT  ',( FOP(I),OPF(I)    DO I=
          1  TO  NOP  ))  SKIP;
         T1=UNSPEC (FOP(1)) ;
         T2=UNSPEC (FOP(2))  ;
         GO TO CONV  ;
   END  ;
 IF   NT(1) & NT(2)  THEN DO;
     DEC :  GET FILE (SYSIN)  LIST (( DOP(I),OPF(I)  DO I= 1 TO
                                  NOP )) ;
    PUT FILE(SYSPRINT) LIST ('DEC   ', ( DOP(I),OPF(I)  DO I= 1 TO
         NOP)) SKIP;
          T1= UNSPEC (DOP(1)) ;
          T2= UNSPEC (DOP(2))  ;
          IF SUBSTR(T1, 61,4)='1101'B
            THEN T1='00001011'B || SUBSTR(T1,5,56);  /* I3 NEG */
            ELSE  T1='00001010'B || SUBSTR(T1,5,56);  /* I3 PSO */
          IF SUBSTR(T2, 61,4)='1101'B
            THEN T2='00001011'B || SUBSTR(T2,5,56);  /* I3 NEG */
            ELSE  T2='00001010'B || SUBSTR(T2,5,56);  /* I3 PSO */
    END  ;
     /*   END  OF  READ IN   OPRAND      */
  CONV  : J=1       ;
    PUT FILE(SYSPRINT) EDIT ('STRINT-1',T1,'STRINT-2',T2)
          (SKIP,X(5),A(10),B(64)) ;
              FLAG(1)= OPF(1) ;
              FLAG(2)=OPF(2) ;
              FLAG(3)= SUBSTR (OPF(1),5,4) ;
              FLAG(4)= SUBSTR (OPF(2),5,4) ;
          DO  I=1 TO 4  BY 2 ;                 /* TEMP(+)= LW  OFA*/
          TEMP(I) =SUBSTR (T1, J, 32)   ;      /* TEMP(2)=LW OF B*/
          TEMP(I+1)=SUBSTR(T2,J,32)     ;      /* TEMP(3)=RW  OF A*/
          J=J+32 ;
          END ;
     /*   SET UP V  &  TRANSMIT  TO  AU  */
  TP_SET_V :  SUBSTR ( V,4,4) = IV_S ;
              NWROD = NOP * 2  ;
              SUBSTR  ( V,8,2) =NT_S ;
   GO TO START;
  TP_SET_NEXT_V  :  IF NOP=2 THEN CT=CT+1;
                            ELSE CT=CT+2;
                    IF  CT <=4          THEN GO TO START ;
                    ELSE  GO TO EXEC_CONT  ;
```

```
START  :     DO  I= 1 TO  4  BY  1 ;
             SUBSTR  (V,I*10+1,8)=  SUBSTR (TEMP(CT),(I-1)*8+1,8);
             SUBSTR  (V,I*10+9,1) = SUBSTR (FLAG(CT),I,1 ) ;
              /*  BIT 10 BE ADDED  LATER  */
             END  START ;   /* END OF DO LOOP  */
  PUT FILE(SYSPRINT) EDIT ('V    ',(SUBSTR(V,(I-1)*10+1,10)
    DO I= 1 TO 5)) (SKIP,A,5 B(11)) ;
CALL TIMER;
AU_SEQ_IN :   IF   WRDCT=0   THEN  DO ;
                                    INDIVR= IV_S ;
                                    NTDNTR= NT_S ;
                                    END  ;
/* FX-POLY */  IF IV(1) & IV(2) & IV(4) & ¬NT(1)
                  THEN GO TO FXPOLY;
/* FL-PLOY */  IF IV(1) & IV(2) & IV(4) &NT(1) & ¬NT(2)
                  THEN GO TO FLPOLY;   /* SUBSEQUENT POLY */
               IF   IV(1)&NT(1)& ¬NT(2)  THEN  GO TO FL_OP_2 ;
                    /*  IF  FL(ADD/SUB/MPY/DIV/CPRA/POLY)
                        THEN GOTO  FL_OP_2 (MOVE V TO REG) */
               IF   IV(1) & IV(2) & ¬IV(3) & ¬IV(4) & ¬NT(1)
                    THEN  GO TO  FXMPY ;   /* FIX MPY  */
               IF    IV(1)& IV(2)&IV(3)& ¬IV(4)& ¬NT(1)
                        THEN GO TO  FXDIV ;  /* FIX DIV */
/* CVD-FX */    IF ¬IV(1) & ¬IV(2) & IV(3) & IV(4) & ¬NT(1)
                    THEN GO TO FX_CVD;
/* CVF-FX */    IF ¬IV(1) & ¬ IV(2) & IV(3) & ¬IV(4) & ¬NT(1)
                  THEN GO TO FX_CVF;
/* CVD-FL  OR CVL FL  */
             IF ( ¬IV(1) & ¬IV(2) & IV(3) &  IV(4)  &NT(1)
                  & ¬NT(2))
               |( ¬IV(1) & ¬IV(2) & ¬IV(3) & IV(4) & NT(1)
                   & ¬NT(2)) THEN GO TO FL_CVD_CVL;
/* CVF-DEC  RO CVL-DEC  */
        IF ( ¬IV(1) & ¬IV(2) & IV(3) & ¬IV(4) & NT(1) & NT(2))
          |( ¬IV(1) & ¬IV(2) & ¬IV(3) & IV(4) & NT(1) & NT(2))
                   THEN GO TO DEC_CVF_CVL;
                                /* THIS IS  FIX CVD/CVF  */
              GO TO ERR ;  /* WRONG  CONTROL CODE  */
FLPOLY : IF ¬POLIP THEN GO TO FL_OP_2;  /* FIRST TIME */
       /* FOR SUBSEQUENT POLY */
       IF WRDCT=0 THEN DO;
          SIGNB=SUBSTR(V,11,1);
          CALL EBDEUM;
          CALL VDUH1;
          GO TO F;   END;
       IF WRDCT=1 THEN DO;
          CALL VDUH2;
          GO TO EXEC_CONT;     END;
  FL_OP_2 :  IF  WRDCT = 0  THEN  DO  ;
                                   SIGNA= SUBSTR (V,11,1) ;
                                   CALL EADEUU ;
                                   CALL FA1LFA ;
                                   CALL  VDUQ1  ;
                                   GO TO F ;  END;
                ELSE  IF  WRDCT=1   THEN  DO ;
                                   SIGNB= SUBSTR(V,11,1)     ;
                                   CALL EBDEUM  ;
```

```
                                    CALL FB1LFB  ;
                                    CALL VDUH1 ;
                                    GO TO F ;   END;
                               ELSE  IF  WRDCT=2 THEN DO;
                                    CALL FA2RFA;
                                    CALL VDUQ2 ;
                                    GO TO  F  ;  END;
                                  ELSE  IF  WRDCT=3 THEN DO;
                                    CALL VDUH2 ;
                                    CALL FB2RFB;
                                    GO TO  EXEC_CONT ;  END;
                                  ELSE  GO TO ERR;
FXPOLY : IF ¬POLIP THEN GO TO FXMPY;  /* FIRST TIME */
        /* FOR SUBSEQUENT POLY */
        CALL VDUH2;
        GO TO EXEC_CONT;
    FXMPY  :   IF  WRDCT=0 THEN DO ;
                                    SIGNA=SUBSTR(V,11,1);
                                    CALL VDUQ2 ;
                                      CALL FA1LFA;
                                    GO TO F;  END;
                    ELSE IF WRDCT=1 THEN DO;
                                    SIGNB=SUBSTR(V,11,1);
                                    CALL FB1LFB;
                                    CALL VDUH3 ;
                                    GO TO  EXEC_CONT ; END ;
                               ELSE   GO TO ERR ;
    FXDIV  :   IF  WRDCT=0 THEN  DO ;
                                   SIGNA=SUBSTR(V,11,1);
                                   CALL  FA1LFA;
                                   CALL VDUQ3 ;
                                   GO TO F;    END ;
                    ELSE  IF WRDCT=1  THEN DO ;
                                   SIGNB=SUBSTR(V,11,1);
                                   CALL FB1LFB;
                                   CALL VDUQ2;
                                   CALL VDUH3  ;
                                   GO TO EXEC_CONT ;  END;
                              ELSE  GO TO ERR  ;
FL_CVD_CVL : IF WRDCT=0 THEN DO;
                                    SIGNA= SUBSTR (V, 11, 1) ;
                                    CALL EBDEUM;
                                    CALL VDUQ1 ;
                                    CALL  FA1LFA;
                                    GO TO F ;   END ;
                    ELSE  IF WRDCT=1  THEN  DO;
                                    CALL VDUQ2 ;
                                    CALL FA2RFA;
                                    GO TO EXEC_CONT  ; END ;
                                  ELSE  GO TO ERR ;
    FX_CVF :                  CALL  VDUQ3;
                              SIGNA=SUBSTR(V,11,1);
                              CALL FA1LFA;
                              GO TO EXEC_CONT ;
FX_CVD : CALL VDUQ2;
                              SIGNA=SUBSTR(V,11,1);
        CALL FA1LFA;
```

```
        GO TO EXEC_CONT;
DEC_CVF_CVL: IF WRDCT=0 THEN DO;
             SIGNA=SUBSTR(V,18,1);
             CALL VDUQ1;
             CALL FA1LFA;
             GO TO F;   END;
             IF WRDCT=1  THEN DO;
             CALL VDUQ2;
             CALL FA2RFA;
             GO TO EXEC_CONT;   END;
   F    :   WRDCT = WRDCT +1 ;
            GO  TO  TP_SET_NEXT_V          ;
    ERR : PUT FILE(SYSPRINT) LIST ('WRONG CONTROL CODE OR NOP')
      PAGE;
           CALL  CLRREG   ;
           GO TO  TPSIM   ;   /*  ERROR  , DO  NEXT  OPERATION */
  /* THIS IS EXECUTION CONTROL PART, CALL VARIOUS ROUTINE  */
   EXEC_CONT :   BEGIN  ;      /* BRANCH TO PROPER ROUTINE */
   PUT FILE(SYSPRINT) EDIT ('SIGNA=',SIGNA,'SIGNB=',SIGNB,
        'EUU=',EUU,'EUM=',EUM,'UH_REG=',UH,'UQ_REG=',UQ,'FA=',FA,
          'FB=',FB)  (SKIP,A(10),B(65));
  /* CVL */       IF  ¬IV(1)& ¬IV(2) & ¬IV(3) &  IV(4)  THEN DO;
                                  CALL TIMER;
                                  CALL CVL;  GO TO CLEAR;  END;
  /* CVF */       IF  ¬IV(1) & ¬IV(2)  &  IV(3) & ¬IV(4)  THEN DO;
                                  CALL TIMER;
                                  CALL CVF;  GO TO CLEAR; END;
  /* CVD */       IF  ¬IV(1) & ¬IV(2) &  IV(3) &  IV(4)  THEN  DO;
                                  CALL TIMER;
                                  CALL CVD;  GO TO CLEAR;  END;
  /* ADDX */      IF   IV(1) & ¬IV(2)  & ¬IV(3)  & ¬IV(4)  THEN DO;
                                  CALL TIMER;
                                  CALL ADDX; GO TO CLEAR;  END;
  /* SUB */       IF   IV(1) & ¬IV(2) &  IV(3) & ¬IV(4)  THEN DO;
                                  CALL TIMER;
                                  CALL SUB;  GO TO CLEAR;  END;
  /* CPRA */      IF   IV(1)& ¬IV(2)  &  IV(3)  &IV(4)  THEN DO;
                                  CALL TIMER;
                                  CALL CPRA;  GO TO CLEAR;  END;
  /* MPY */        IF   IV(1) & IV(2) & ¬IV(3) & ¬IV(4) THEN DO;
                                  CALL TIMER;
                                  CALL MPY;  GO TO CLEAR;  END;
  /* POLYX */      IF   IV(1) & IV(2)  & ¬IV(3) &  IV(4)THEN DO;
                                  CALL TIMER;
                                  CALL POLYX; GO TO KLEAR;  END;
  /* DIV */        IF   IV(1) & IV(2) & IV(3) & ¬IV(4)  THEN    DO;
                                  CALL TIMER;
                                  CALL DIV;  GO TO CLEAR;  END;
  /* POLYX */      IF   IV(1) & IV(2) & IV(3) & IV(4)   THEN    DO;
                                  CALL TIMER;
                                  CALL POLYX; GO TO KLEAR;  END;
                  ELSE GO TO ERR  ;
      END  EXEC_CONT  ;
   FA1LFA  : PROCEDURE   ;  /* V(19,29,39,49) = FA(1- 4)        */
             /*  FLAG OF AL TO LEFT PART OF  FA REG  */
             DO  I=1  TO  4  BY  1  ;
             SUBSTR (FA,I,1) = SUBSTR (V,I*10+9, 1 )  ;
```

```
            END ;
            RETURN ;  END FA1LFA ;
FA2RFA   : PROCEDURE   ;  /* V(19,29,39,49) = FA(5-8)          */
            /* FLAG OF A2 TO RIGHT PART OF FA REG */
           DO I=1  TO  4  BY  1  ;
            SUBSTR (FA, I+4 ,1)=SUBSTR (V, I*10+9 , 1 ) ;
            END ;
            RETURN;  END ;
FB1LFB   : PROCEDURE ;  /* V(19,29,39,49) =FB(1-4) */
            /* FLAG OF B1  TO LEFT PART OF FB  REG  */
            DO I=1  TO  4  BY  1  ;
            SUBSTR (FB,I, 1) = SUBSTR (V, 10*I+9 , 1  );
   END ;
            RETURN ; END  FB1LFB ;
FB2RFB   : PROCEDURE  ;  /* V(19,29,39,49)=FB(5-8) */
            /* FLAG OF B2  TO LEFT PART  OF FB REG  */
            DO  I=1  TO  4  BY  1  ;
            SUBSTR  (FB, I+4, 1)=SUBSTR (V, 10*I+9 , 1) ;
            END ;
            RETURN ; END FB2RFB ;
EADEUU : PROCEDURE   ;      /*  V BIT(12-18) = EUU (BIT 1-7)  */
            EUU= SUBSTR (V , 12, 7)  ;
            RETURN;  END  EADEUU ;     /* EXP OF A-OP  TO EUU   */
EBDEUM   : PROCEDURE   ;      /*  V (BIT 12-18) =EUM(1-7)       */
            EUM = SUBSTR (V , 12, 7)  ;
            /* EXP OF  B-OP  TO EUM REG  */
           RETURN ;  END  ;
END  AUSIM;
/*
```

```
//              EXEC    PL1
//PL1.SYSIN     DD    *
      GATE : PROCEDURE ;
     DCL   MEPB BIT(4) PACKED EXT;
     DCL M BIT(64) PACKED EXT,  (LQ,UQ) BIT(65) PACKED EXT,  II EXT,
          (UH,LH,US,LS,UM,LM,X,Y,S,T,Z,N,G)  BIT(64) PACKED EXT,
       (P,B) BIT(64) PACKED EXT,
          (OV,UN,LSG,ID,GT,EQ,LT,FM,BOGUS) BIT(1) EXT,
      (EUU,EUM,EUL) BIT(7) EXTERNAL,
          (FA,FB) BIT(8)  EXTERNAL,
          V  BIT(50)  EXTERNAL;
     DCL NEGR EXT;
     VDM1    :  ENTRY        ;
           /*  M.BYTE (1-3) = V.BYTE (3-5)  BIT  1-8    */
             DO  I=1    TO  3  BY  1  ;
             SUBSTR(M,(I-1)*8+9,8) =SUBSTR(V,I*10+11,8) ;
             END ;
               RETURN;
     VDM2    :ENTRY       ;
             /* M.BYTE(4-7)= V.BYTE(2-4) BIT 1-8  */
             DO I=1    TO  3  BY  1  ;
             SUBSTR(M,I*8+25,    8) =SUBSTR(V,I*10+1, 8) ;
             END;
             RETURN;
     VDUQ1   : ENTRY;
            /*  UQ.BYTT(1-3)= V.BYTE(2-4) BIT 1-8 */
             DO  I=1  TO  3  BY  1  ;
             SUBSTR (UQ, I*8+1, 8)=SUBSTR (V,I*10+11,8);
             END ;
             RETURN;
     VDUH1   :ENTRY;
             /*  UH.BYTE(1-3)=V.BYTE(2-4) BIT1-8 */
             DO I=1  TO  3  BY  1  ;
             SUBSTR (UH , I*8+1 , 8) = SUBSTR (V,I*10+11,8)  ;
             END;
             RETURN;
     VDUQ2   : ENTRY;
             /* UQ.BYTE(4-7)= V.BYTE(1-4) BIT 1-8  */
             DO  I=1  TO  4  BY  1  ;
             SUBSTR (UQ,I*8+25,8)=SUBSTR (V,I*10+1,8);
             END;
             RETURN;
     VDUH2   : ENTRY;
             /*  UH. BYTE (4-7) =V.BYTE(1-4) BIT 1-8 */
             DO  I=1  TO  4  BY  1  ;
             SUBSTR  (UH,I*8+25,8)=SUBSTR  (V,I*10+1,8);
             END;
             RETURN;
     VDUQ3   : ENTRY;
             /* UQ.BYTE(0-3)= V.BYTE (1-4) BIT 1-8 */
             DO I=1   TO  4  BY  1  ;
             SUBSTR (UQ, (I-1)*8+1, 8) = SUBSTR (V, I*10+1, 8);
             END;
             RETURN;
     VDUH3   : ENTRY;
             /* UH.BYTE(0-3))= V.BYTE(1-4) BIT 1-8 */
             DO I=1  TO  4  BY  1  ;
```

```
        SUBSTR(UH, (I-1)*8+1, 8) = SUBSTR(V, I*10+1, 8);
        END;
        RETURN;
ML7Y1   :ENTRY;
        Y   = (64)'0'B    ;  /*  CLEAR Y  */
        SUBSTR(Y,1,57)=SUBSTR(M,8,57) ;
        RETURN;
ML6Y1   : ENTRY;
        Y   = (64)'0'B    ;  /*  CLEAR Y  */
        SUBSTR(Y,1,58)=SUBSTR(M,7,58) ;
        RETURN;
UHDY1 : ENTRY;
        Y=UH;
        RETURN;
MDY1    : MDY4 :  ENTRY;
        Y   = (64)'0'B    ;  /*  CLEAR Y  */
        Y=M;
        RETURN;
ML5Y2   : ENTRY;
        Y   = (64)'0'B    ;  /*  CLEAR Y  */
        SUBSTR(Y,1,59)=SUBSTR(M,6,59) ;
        RETURN;
ML4Y2   :  ENTRY;
        Y   = (64)'0'B    ;  /*  CLEAR Y  */
        SUBSTR(Y,1,60)=SUBSTR(M,5,60);
        RETURN;
ML3Y3   :  ENTRY;
        Y   = (64)'0'B    ;  /*  CLEAR Y  */
        SUBSTR(Y,1,61)=SUBSTR(M,4,61) ;
        RETURN;
ML2Y3   :  ENTRY;
        Y   = (64)'0'B    ;  /*  CLEAR Y  */
        SUBSTR(Y,1,62)=SUBSTR(M,3,62) ;
        RETURN;
ML1Y4   :  ENTRY;
        Y   = (64)'0'B    ;  /*  CLEAR Y  */
        SUBSTR(Y,1,63)=SUBSTR(M,2,63) ;
        RETURN;
PDY4    : ENTRY;
        Y=P;
        RETURN;
LMDM  : ENTRY;
        M=LM;          /*  M (BYTE 0-7) = LM(BYTE /-7) */
        RETURN;
UHDM  : ENTRY;
        M=UH;          /*  M (BYTE 0-7) = UH(BYTE 0-7) */
        RETURN;
LSL8US : ENTRY;
  US=(64)'0'B;  SUBSTR(US,1,56)=SUBSTR(LS,9,56);
        RETURN;
UHDUS : ENTRY;
        US = UH;
        RETURN;
LSR8US : ENTRY;
  US=(64)'0'B;   SUBSTR(US,9,56)=SUBSTR(LS,1,56);
        RETURN;
LML8UM : ENTRY;
```

```
        UM=(64)'0'B;    SUBSTR(UM,1,56)=SUBSTR(LM,9,56);
                RETURN;
      LMR8UM   : ENTRY;
        UM=(64)'0'B;    SUBSTR(UM,9,56)=SUBSTR(LM,1,56);
                RETURN;
       UQDUM   :ENTRY;
                UM = UQ   ;
                RETURN;
     LHL4UH : ENTRY;
        UH=(64)'0'B;    SUBSTR(UH,1,60)=SUBSTR(LH,5,60);
                RETURN;
     LHR4UH : ENTRY;
        UH=(64)'0'B;    SUBSTR(UH, 5,60)=SUBSTR(LH,1,60);
                RETURN;
     LHL8UH : ENTRY;
        UH=(64)'0'B;    SUBSTR(UH,1,56)=SUBSTR(LH,9,56);
                RETURN;
     LHR8UH : ENTRY;
        UH=(64)'0'B;    SUBSTR(UH, 9,56)=SUBSTR(LH,1,56);
                RETURN;
     LSDUH   : ENTRY;
                UH = LS ;
                RETURN;
     UHDLH   : ENTRY;
                LH=UH;
                RETURN;
     UQDLQ   : ENTRY;
                LQ=UQ;
                RETURN;
     UMDX1 : ENTRY;
                X = UM   ;
                RETURN;
   USDS1 : ENTRY;
                S = US ;
                RETURN;
LQL4UQ : ENTRY;
     UQ=(65)'0'B;
     UQ=SUBSTR(LQ,5,61) || '0000'B;
    RETURN;
   LQR4UQ   : ENTRY;
    UQ=(65)'0'B;
    SUBSTR(UQ,5,61) = SUBSTR(LQ,1,61);
                RETURN;
LMDUQ : ENTRY;
    UQ=(65)'0'B;
    SUBSTR(UQ,1,64)=LM;       RETURN;
LQR8UQ : ENTRY ;
    UQ=(65)'0'B;
    SUBSTR(UQ,9,57)=SUBSTR(LQ,1,57);
    RETURN;
LQL8UQ : ENTRY;
     UQ=(65)'0'B;       UQ=SUBSTR(LQ,9,57) || '00000000'B;
    RETURN;
       Z4DLM : ENTRY;
              LM = Z;
              RETURN;
T4DLS: ENTRY;
```

```
   LS=T;
   RETURN;
  LSDUS : ENTRY;
      US=LS;
      RETURN;
  LMDUM : ENTRY;
      UM=LM;
      RETURN;
  LHDUH : ENTRY;
      UH=LH;
      RETURN;
  LQDUQ : ENTRY ;
      UQ=LQ;
      RETURN;
ASSIM : ENTRY;
    G=(64)'1'B;
        CALL  SDS  (X,Y,S,G,N,T,Z)  ;        /*SDS  ADDER */
        CALL  PROP  ;                        /*  PROPOGATION LOGIC*/
   Y=(64)'0'B ;   N=Y;   G=Y;
   CALL  SDS (Z,Y,T,G,N,T,Z)  ;              /* PASSING SDS2 */
   CALL  SDS (Z,Y,T,G,N,T,Z)  ;              /* PASS SDS3 */
   P=B ;   CALL PDY4 ;
   CALL  SDS (Z,Y,T,G,N,T,Z) ;               /*  Z= ASSIMILATEOUT*/
    RETURN;
NORM : ENTRY;                                                    */
      /*  II = EUL EXTENDED TO 16 BITS
   IF  SUBSTR(UQ,5,4) ¬='0000'B  THEN DO;
       CALL UQDLQ;
       CALL LQR4UQ;
       II=II+1;
    IF II >  63  THEN OV='1'B;  /*  EUL > 63  */
      GO TO EXPR;
      END;
TEST8 : IF SUBSTR(UQ,9,4) ¬= '0000'B  THEN GO TO EXPR;
       IF SUBSTR(UQ,13,4) ¬= '0000'B  THEN GO TO LL4;
         /* 8 LEADING ZEROES  */
         CALL UQDLQ;
         CALL LQL8UQ;
         II=II-2;
         GO TO TEST8;
LL4 : CALL UQDLQ;            /* 4 LEADING ZEROES  */
       CALL LQL4UQ;
       II=II-1;
       IF II <-64 THEN UN='1'B;  /* EUL < -64  */
EXPR : EUL=SUBSTR( UNSPEC(II+64), 26, 7);
    CALL SETBOG;
   RETURN;
SETBOG : ENTRY;
     IF OV THEN EUL=(7)'1'B;
     IF UN THEN EUL=(7)'0'B;
     BOGUS=OV|UN|LSG|ID;
     IF BOGUS THEN CALL INDFA;
     RETURN;
INDFA : ENTRY;
     FA=(8)'0'B;
     SUBSTR(FA,1,1)= GT;
     SUBSTR(FA,2,1)= EQ;
```

```
     SUBSTR(FA,3,1)= LT;
     SUBSTR(FA,4,1)= OV;
     SUBSTR(FA,5,1)= FM;
     SUBSTR(FA,6,1)= UN;
     SUBSTR(FA,7,1)= LSG ;
     SUBSTR(FA,8,1)= ID;
     RETURN;
TIMER : ENTRY;
        DCL TAME CHARACTER(9);
        TAME=TIME;
        PUT FILE(SYSPRINT) EDIT ('TIME= ',SUBSTR(TAME,1,2),' . ',
          SUBSTR(TAME,3,2), ' . ', SUBSTR(TAME,5,2), ' . ',
          SUBSTR(TAME,7,3)) (SKIP,X(40),A,A(2),A,A(2),A,A(2),A,A(3));
  END GATE;
*
/             EXEC PL1
//PL1.SYSIN    DD *
ARITH : PROCEDURE;
MPY : DIV : CVL : CVF : CVD : ENTRY; RETURN;
POLYX : ENTRY;
   RETURN;
   END ARITH;
*
```

```
//              EXEC   PL1
//PL1.SYSIN     DD  *
       SDS    :  PROCEDURE  ( X,Y,S,G,N,T,Z);
                 DCL     (X,Y,S) BIT (64),     N  BIT (64),
                         (G,T,Z)  BIT (64);
                 DCL    (C,CC)BIT (64);
     DCL (C2,C3) BIT(1);
   DCL  C1 BIT(1);
      DCL   PRINT FIXED BIN EXT;
            /*  S = SIGN OF MINUEND IN SDS FORMAT          */
            /*  X = MAGNITUDE OF MINUEND IN SDS FORMAT     */
            /*  Y=  SUBTAHEND IN  CONUENTIONAL FORMAT      */
            /*  T=  SIGN OF RESULT IN SDS FORMAT           */
            /*  Z=  MAGNITUDE OF RESULT IN SDS FORMAT      */
            /*  N = NEGATION CONTROL                       */
            /*  G = GATE ON INTERSTAGE CONNECTS            */
            /*  C = INTERSTAGE CONNECTS                    */
            /* CC=  TEMPORARY STORAGE                      */
          IF PRINT=0  THEN GO TO SDSC;
          PUT FILE(SYSPRINT) LIST ('INPUT TO SDS') SKIP(2);
         PUT FILE(SYSPRINT) EDIT ('S',S,'X',X,'Y',Y,'N',N,'G',G)
          (SKIP,A(5),X(4),B(64));
   SDSC :     CC=S & X;     C2=SUBSTR(CC,1,1);
              CC= (CC|( ¬X & Y)) & G;
               C=(64)'0'B                ;
               SUBSTR (C,1,63) =SUBSTR( CC,2,63)    ;
             /* CC.I-1 = X.I S.I + X.I Y.I) G.I,    C.I==  CC.I+1  */
               CC = (X &  ¬Y ) | (¬X & Y)     ;      /* CC = X .XOR.Y
               Z =   (C &  ¬CC) |(¬C & CC ) ;        /* Z= CC.XOR.C Y
               T =  (C &   ¬N) | (¬C & N )   ;       /*  = C .XOR. N
     IF C2 THEN DO;
     C1=SUBSTR(T,1,1);  SUBSTR(T,1,1)=¬C1;
     PUT FILE(SYSPRINT) DATA (C1,C2);   END;
         IF PRINT=0 THEN GO TO OUTX;
         PUT FILE(SYSPRINT) LIST ('OUTPUT OF SDS') SKIP;
        PUT FILE(SYSPRINT) EDIT ('T',T,'Z',Z) (SKIP,A(5),X(4),B(64))
   OUTX :      RETURN ;  END SDS  ;
/*
```

```
//              EXEC PL1
//PL1.SYSPUNCH DD  UNIT=PUNCH,DSNAME=&PUNCH
//PL1.SYSIN    DD *
       PROP  :  PROCEDURE              ;  /* PARAMETERS ARE B,T,Z  */
     DCL (B,T,Z) BIT(64) PACKED EXT;
    DCL  (BC,BB,TT, ZZ)   BIT(1)  ;
            /*  B= PROPAGATION BIT                */
            /*  T= SIGN                           */
            /*  Z= MAGNITUDE                      */
            /*  BB= TEMPORARY STORAGE             */
              SUBSTR (B,64,1) = '0'B  ;     /* BIT 64 OF B=0  */
              DO    I= 63  TO 1  BY  -1     ;
              BC = SUBSTR (B,I,1)     ;
              BB = SUBSTR (B,I+1,1)  ;
              TT = SUBSTR (T,I+1,1)  ;
              ZZ = SUBSTR (Z,I+1,1)  ;
              SUBSTR (B,I,1) = BB & ¬ZZ | TT & ZZ  ;
               /*  B.I = B.I+1 &  ¬Z.I+1  |  T.I+1  & Z.I+1  */
              END     ;
  PUT  FILE (SYSPRINT) DATA ( T,Z,B)   SKIP(2);
  END PROP;
/*
```

```
//             1142,DCS,08.00,150,900),RMLANSFORD,MSGLEVEL=1
//             EXEC  PL1
//PL1.SYSPUNCH DD  UNIT=PUNCH,DSNAME=&PUNCH
//PL1.SYSIN    DD  *
    ADDX : PROCEDURE;
      DCL (M,UH,LH,US,LS,UM,LM,X,Y,S,T,Z,N,G) BIT(64)  PACKED EXT,
          (LQ,UQ) BIT(65) PACKED EXT,  II EXT,
          (P,B) BIT(64) PACKED EXT,
           (OV,UN,LSG,ID,GT,EQ,LT,FM,BOGUS) BIT(1) EXT,
           (EUU,EUM) BIT(7) EXTERNAL,  EUL  BIT(7) EXTERNAL,
           (FA,FB)  BIT(8) EXTERNAL,
           (SIGNA,SIGNB) BIT(1) EXTERNAL, POLIP BIT(1) EXTERNAL,
           V BIT(50) EXTERNAL,
           (SDB,SR) BIT(1) EXTERNAL,
           AEXPGT  FIXED BIN EXTERNAL;
       DCL    IOP(2) FIXED BIN(31) EXTERNAL,
              FOP(2)  FLOAT BIN(53)  EXTERNAL,
              DOP(2)  DECIMAL FIXED (15) EXTERNAL;
      DCL     DEXP FIXED BIN,      CI FIXED BIN;
      DCL NEGO BIT(64) , (TP,TO,T1) BIT(1);
       DCL    (CPR,MINUS,PLUS) BIT(1);
       DCL FP BIT(8);
       DCL    (DIFF,TEMP) BIT(64);
       ON OVERFLOW OV='1'B;  ON UNDERFLOW UN='1'B;
       FOP(1)=FOP(1)+FOP(2);
       PLUS='1'B;  CPR,MINUS='0'B;
       GO TO FIRST;
     SUB : ENTRY;
       ON OVERFLOW OV='1'B;  ON UNDERFLOW UN='1'B;
       MINUS='1'B;  CPR,PLUS='0'B;     /* SUB INDICATOR */
       FOP(1)=FOP(1)-FOP(2);
       GO TO FIRST;
     CPRA : ENTRY;
       CPR='1'B;  MINUS,PLUS='0'B;
     FIRST : TEMP=UNSPEC(FOP(1));
       PUT FILE(SYSPRINT) EDIT ('RESULT=',FOP(1),TEMP)
           (SKIP,A(7),E(14,6),X(2),B(64));
      DEXP=EUU-EUM ;      PUT FILE(SYSPRINT) DATA (DEXP);
      IF DEXP > 0  THEN  GO TO  EXPGTO ;
      IF DEXP = 0  THEN  GO TO  EXPEQO ;
      ELSE  GO TO  EXPLTO ;                          /*  EXP >0  */
     EXPGTO :  AEXPGT = 1 ;
               SDB='1'B;   SR=SIGNA;
               IF CPR THEN GO TO SETFLAG;
               EUL = EUU  ;
               IF DEXP > 13  THEN  GO TO  EXPGT13 ;
               ELSE GO TO  RSUH  ;
      EXPGT13 : GO TO  ADDEND;
      RSUH    : IF  DEXP >= 2  THEN  DO  ;
                    CALL UHDLH ;  CALL LHR8UH ;  DEXP=DEXP-2 ;
                    GO TO RSUH  ;  END;
                ELSE  IF  DEXP=0  THEN  GO TO  CAL  ;
                    ELSE  CALL  UHDLH ;  CALL LHR4UH ;
                    GO  TO   CAL  ;
      EXPEQO  : EUL=EUU  ;  AEXPGT=1;
                /*   CPRA WITH SAME EXP BUT DIFFERENT SIGN          */
                /*   OP. A + GT;   OP. A -   LT                     */
```

```
            IF SUBSTR(V,4,4)='1000'B  THEN SDB=SIGNA=SIGNB;
            ELSE SDB=( SIGNA & ¬SIGNB)|( ¬SIGNA & SIGNB);
            IF SDB THEN DO;
              SR=SIGNA;
              IF CPR THEN GO TO SETFLAG;
            END;
            GO TO CAL;
  EXPLTO : EUL=EUM;
            SDB='1'B;     SR=(SIGNB & ¬MINUS)|( ¬SIGNB & MINUS);
            /*   SR =SIGNB .XOR. SUB                                  */
            IF CPR THEN GO TO SETFLAG;
            IF   DEXP  <  (-13)  THEN  DO  ;
                 CALL  UHDM ;
                 X=(64)'0'B ;  S=X;   G=X ;   N=X;
                 CALL  MDY4  ;
                 CALL  SDS (X,Y,S,G,N,T,Z) ;
                 CALL  Z4DLM ; CALL LMDUQ ;
                 GO TO ADDEND;  END;
            ELSE
       RSUQ  :  IF  DEXP  <= (-2)  THEN  DO ;
                    CALL     UQDLQ;  CALL  LQR8UQ ;
                    DEXP =DEXP+2 ;
                    GO TO RSUQ ;  END ;
                ELSE  IF  DEXP =0  THEN GO TO CAL;
                      ELSE  CALL UQDLQ;   CALL LQR4UQ;
                      GO TO  CAL  ;
  CAL :  PUT FILE(SYSPRINT)  EDIT  ('AFTER SHIFT','UQ-REG',UQ,
         'UH_REQ',UH)                (SKIP,A(12),       SKIP,A(12),
         B(64),SKIP,A(12),B(64));
    CALL UHDM;   CALL UQDUM ;
  /*  GET    SDB  AND  NEG1  (N=NEG1) */
  IF  SUBSTR (V,4,4)='1000'B THEN DO  ;          /* FOR ADD */
      /*  NEGO= SIGNA=SIGNB                                          */
      /*  NEG1=(SIGNA=SIGNB)|(SIGNB=SR)                              */
       TO = SIGNA = SIGNB ;
       T1=SIGNB=SR;      END;
  ELSE  DO ;                                    /* FOR  SUB & CPRA */
      /*  NEGO= SIGNA .XOR. SIGNB                                    */
      /*  NEG1=(SIGNA .XOR.SIGNB)|( SIGNB .XOR. SR)                  */
       TO=(SIGNA & ¬SIGNB) | ( ¬SIGNA & SIGNB) ;
       T1=( SIGNB & ¬SR)|( ¬SIGNB & SR);       END;
  IF    TO='0'B   THEN  NEGO =(64)'0'B ; ELSE  NEGO=(64)'1'B ;
   T1=T1 | TO;
  IF    T1  THEN N=(64)'1'B;     ELSE  N=(64)'0'B;
  US= (US & ¬NEGO) | ( ¬ US  &  NEGO) ;      /*   US.XOR. NEGO */
  CALL UMDX1 ;   CALL USDS1;  CALL MDY1 ;
    PUT FILE(SYSPRINT) DATA (SDB);
   PUT FILE(SYSPRINT) LIST ('ASSIM  BEGIN') SKIP(2);
D1 :  CALL ASSIM;
   IF  SDB='1'B  THEN  DO ;
               CALL Z4DLM;  GO TO TERM;  END;
  ELSE SR=SUBSTR(Z,7,1);
   IF     SR='0'B   THEN  DO ;
      CALL  Z4DLM ;  GO TO  TERM ; END;
   ELSE  DO  ;
            T1= ¬T1;   SDB='1'B;
      IF   T1='0'B THEN N=(64)'0'B;  ELSE N=(64)'1'B;
```

```
      CALL UMDX1;  CALL USDS1;  CALL MDY1;
         GO TO  D1 ;    END;
   TERM :   CALL  LMDUQ ;
            IF UQ=(64)'0'B  THEN EQ='1'B;
    IF  SUBSTR (V,4,4)='1011'B   THEN  GO TO SETFLAG;  /* FOR CPRA */
    II=EUL-64;      /*  II= EUL EXTENDED TO 16 BITS              */
     IF ¬EQ  THEN GO TO NORMAL;
        IF EUL ¬=(7)'0'B  THEN DO;     /* UQ=0,EUL ¬= -64       */
        LSG='1'B;  CALL INDFA;  GO TO ADDEND;  END;
  NORMAL : CALL NORM;
      GO TO ADDEND;
  SETFLAG : FP=( FA & ¬FB) |( ¬FA & FB);
            IF FP ¬= (8)'0'B THEN FM='1'B;
            GT= ¬SR & ¬EQ;
            LT= SR & ¬EQ;
            CALL INDFA;
  ADDEND :  PUT  FILE (SYSPRINT) DATA ( SR,EUL,UQ,FA,OV,UN) SKIP(2);
        TEMP=UNSPEC(FOP(1)); SUBSTR(TEMP,1,8)=(8)'0'B;
       DIFF=(UQ & ¬TEMP)|( ¬UQ & TEMP);
        PUT FILE(SYSPRINT) EDIT ('DIFF=',DIFF) (SKIP,A,B(64));
  END    ADDX ;
/*
```

```
//             1235,DCS,03.00,100,900),'D. E. ATKINS',MSGLEVEL=1
//             EXEC PL1
//PL1.SYSPUNCH DD  UNIT=PUNCH,DSNAME=&PUNCH
//PL1.SYSIN    DD *
   MPY : PROCEDURE;
      /* .                                                          */
      /* . THIS ROUTINE INCLUDS  FXMPY AND FLMPY                    */
      /* . NCC= NO. OF  MPY CYCLE                                   */
      /* .       2  FOR  SH FX                                      */
      /* .       4  FOR  LG FX                                      */
      /* .       7  FOR  FL                                         */
      /* . AT END OF MPY CYCLES, EACH NUMBER TYPE                   */
      /* .    LOAD DIFFERENTLY INTO REGISTER                        */
      /* . THEN  ASSIMILATION                                       */
      /* . ONLY  FL MPY NEEDS NORMALIZATION                         */
      /* .                                                          */
      DCL MEPB BIT(4) PACKED EXT;
      DCL  M BIT(64) EXT, (LQ,UQ) BIT(65) PACKED EXT,   II EXT,
           (UH,LH,US,LS,UM,LM,X,Y,S,T,Z,N,G) BIT(64) PACKED EXT,
        (P,B) BIT(64) PACKED EXT,
           (OV,UN,LSG,ID,GT,EQ,LT,FM,BOGUS) BIT(1) EXT,
           (EUU,EUM) BIT(7) EXTERNAL,  EUL  BIT(7) EXTERNAL,
           (FA,FB)  BIT(8) EXTERNAL,
           (SIGNA,SIGNB) BIT(1) EXTERNAL, POLIP BIT(1) EXTERNAL,
           V BIT(50) EXTERNAL,
           (SDB,SR) BIT(1) EXTERNAL,
           NCC EXT,
           AEXPGT  FIXED BIN EXTERNAL;
       DCL   IOP(2) FIXED BIN(31) EXTERNAL,
             FOP(2)  FLOAT BIN(53)  EXTERNAL,
             DOP(2)  DECIMAL FIXED (15) EXTERNAL,
             OPF(2)  BIT(8)    , FLAG(4) BIT(4) ;
      DCL  (MY128X,MY64X,MY32X,MY16X,MY8X,MY4X,MY2X,MY1X) BIT(1),
           (NO,N1,N2,N3,N4) BIT(1),
           (Q57,Q58,Q59,Q60,Q61,Q62,Q63,Q64,Q65) BIT(1);
  DCL        NEGO BIT(64) ;
      DCL (TEMP, DIFF) BIT(64);
      DCL NT BIT(2), SHALF BIT(16), LHALF  BIT(32);
       PUT FILE(SYSPRINT) EDIT ('START MPY') (SKIP,X(40),A);
       CALL TIMER;
       ON OVERFLOW OV='1'B;   ON UNDERFLOW UN='1'B;
       SR= (SIGNA & ¬SIGNB)|( ¬SIGNA & SIGNB);
      NT= SUBSTR(V,8,2);
      IF  NT='10'B THEN GO TO  MPYFL;
      /* .                                                          */
      /* . THIS PART FOR FIX ONLY                                   */
      /* .                                                          */
  FIXMPY : CALL UHDLH;
      CALL LHR8UH;
       IF SIGNB THEN SUBSTR(UH,1,8)=(8)'1'B;
      PUT FILE(SYSPRINT) EDIT ('UH AFTER SHIFT= ',UH)
               (SKIP,A,B(65));
      CALL UHDM;
      IF NT='00'B THEN NCC=2;
      ELSE  NCC=4 ;
      GO TO  MULT;
       /* ..........                          .............. */
```

```
    /* . THIS PART FOR FLOATING NO. ONLY                          */
    /* .............                                  ..........  */
    /* .............
MPYFL : IF  UH=(64)'0'B|UQ=(65)'0'B THEN DO;
     EUL=(7)'0'B;  SR='0'B;  UQ=(65)'0'B;  GO TO MPYEND;  END;
     II=EUU+EUM-128; /*  II= SUM OF EXPONENT  */
     IF II > 63 THEN OV='1'B;   /*  EUL > 63  */
     IF II <-64 THEN UN='1'B;     /* EUL < -64  */
     CALL UHDM;
    NCC=7;                                                      */
    /* .                                                        */
    /* .THIS PART IS COMMON TO BOTH FX AND FL                   */
    /* .
MULT :  ICC=0 ;
 LOOP :  IF ICC >= NCC  THEN GO TO  MULTEND;
 RECORDER :  BEGIN;
    Q57=SUBSTR(UQ,57,1);    Q58=SUBSTR(UQ,58,1);
    Q59=SUBSTR(UQ,59,1);    Q60=SUBSTR(UQ,60,1);
    Q61=SUBSTR(UQ,61,1);    Q62=SUBSTR(UQ,62,1);
    Q63=SUBSTR(UQ,63,1);    Q64=SUBSTR(UQ,64,1);
    Q65=SUBSTR(UQ,65,1);
    MY128X=( ¬Q57 & Q58 & Q59) | ( Q57 & ¬Q58 & ¬Q59);
    MY64X =( ¬Q58 & Q59) | ( Q58& ¬Q59) ;
    MY32X =( ¬Q59 & Q60 & Q61) | ( Q59 & ¬Q60 & ¬Q61);
    MY16X =( ¬Q60 & Q61) | ( Q60 & ¬Q61);
    MY8X  =( ¬Q61 & Q62 & Q63) | ( Q61 &¬Q62 & ¬Q63);
    MY4X  =( ¬Q62 & Q63) | ( Q62 & ¬Q63);
    MY2X  =( ¬Q63 & Q64 & Q65) | ( Q63 & ¬Q64 & ¬Q65);
    MY1X  =( ¬Q64 & Q65) | ( Q64& ¬Q65);
    N0 = ¬Q57     ;
    N1 = ( Q57 & ¬Q59) | ( ¬Q57 & Q59) ;
    N2 = ( Q59 & ¬Q61) | ( ¬Q59 & Q61) ;
    N3 = ( Q61 & ¬Q63) | ( ¬Q61 & Q63) ;
    N4 = ¬Q63;
    END RECORDER ;
     PUT FILE(SYSPRINT) LIST ('CYCLE COUNT=',ICC,'UQ=',UQ) SKIP(2);
     PUT FILE(SYSPRINT) DATA (MY128X,MY64X,MY32X,MY16X,MY8X,MY4X,MY2
        ,MY1X,N0,N1,N2,N3,N4);
     CALL UQDLQ;
     G=(64)'1'B;
  SDS1 :  CALL UMDX1;
    IF N0='0'B THEN  NEG0=(64)'0'B ;  ELSE  NEG0 =(64)'1'B;
     US = (US & ¬NEG0)|( ¬US &NEG0);
     CALL USDS1;
     Y=(64)'0'B;
     IF MY128X  THEN DO;
          CALL ML7Y1;  GO TO SH1;  END;
     IF MY64X THEN DO;  CALL ML6Y1;  END;
   SH1: IF  N1='0'B  THEN N=(64)'0'B;  ELSE N=(64)'1'B;
        CALL SDS(X,Y,S,G,N,T,Z);
  SDS2:   Y=(64)'0'B;
     IF MY32X THEN DO;  CALL ML5Y2;  GO TO SH2;  END;
     IF MY16X  THEN DO;  CALL ML4Y2;  END;
   SH2: IF N2='0'B  THEN  N=(64)'0'B;  ELSE N=(64)'1'B;
        CALL SDS (Z,Y,T,G,N,T,Z);
  SDS3 : Y=(64)'0'B;
      IF MY8X THEN DO;  CALL ML3Y3;  GO TO SH3;  END;
     IF MY4X THEN DO;  CALL ML2Y3;  END;
```

```
  SH3 : IF N3='0'B  THEN  N=(64)'0'B;  ELSE  N=(64)'1'B;
        CALL  SDS(Z,Y,T,G,N,T,Z);
SDS4 : Y=(64)'0'B;
    IF MY2X  THEN DO;  CALL ML1Y4;  GO TO SH4;  END;
    IF MY1X  THEN DO;  CALL MDY4;  END;
  SH4:  IF N4='0'B THEN N=(64)'0'B ; ELSE N=(64)'1'B;
        CALL  SDS(Z,Y,T,G,N,T,Z);
   CALL T4DLS;
   CALL Z4DLM;
   ICC=ICC+1;
   /* .                                                    */
   /* . AT END OF LAST CYCLE FOR FX NO.                    */
   /* . LOAD REGISTERS BEFORE  ASSIMILATION                */
   /* .                                                    */
    IF (ICC=NCC) & NT='01'B THEN GO TO MULTEND;  /*LFX */
   CALL LQR8UQ;
   CALL LSR8US;
   CALL LMR8UM;
   IF  ICC=7 THEN CALL MEXTPRE;
   GO TO LOOP;
MULTEND : IF NT='01'B THEN DO;      /* FOR LFX ONLY      */
          CALL  LSDUS;
          CALL  LMDUM;  END;
  /*   SET X,Y,S,G,N  TO SDS1  FOR ASSIM   */
   /*.
   /*. BEFORE ASSIM, TEST UQ BIT 65, IF EQUALS1, DO ONE*/
   /*. MORE CYCLE, FOR FL USE MDY1,                     */
   /*.                 LFX USE  UHDY1,                  */
   /*.                 SFX USE  MDY1 ,                  */
   /*.                                                  */
       Y,NEGO,N=(64)'0'B;          /*  N=NEG1          */
   IF NT ¬='10'B THEN GO TO ASS;  /* ONLY FL TEST UQ65 */
TUQ65 : IF SUBSTR(UQ,65,1)='0'B THEN GO TO ASS;  /*UQ65=0 */
   N,NEGO=(64)'1'B;   /* FOR UQ65=1 & FL */
    CALL MDY1;
ASS :  US= (US & ¬NEGO)|( ¬US & NEGO) ;
    CALL USDS1;
    CALL UMDX1;
   PUT FILE(SYSPRINT) LIST ('ASSIM BEGIN')  SKIP(2);
    CALL ASSIM;
    CALL Z4DLM;
    CALL LMDUQ;
   IF NT='10'B THEN GO TO FLEND;
   IF NT='00'B THEN DO;
       CALL  UQDLQ;
       CALL  LQR8UQ;
       SHALF= SUBSTR(UQ,33,16);
       IF  ( ¬SR & SHALF ¬=(16)'0'B)|( SR & SHALF  ¬=(16)'1'B)
            THEN OV='1'B;
       GO TO  FXEND ;
       END;
   ELSE DO;
       LHALF=SUBSTR(UQ,1,32);
       IF ( ¬SR & LHALF ¬=(32)'0'B)|( SR & LHALF ¬=(32)'1'B)
            THEN OV='1'B;
       END;
FXEND :  IOP(1)=IOP(1) *IOP(2);
```

```
    TEMP=(64)'0'B;
    TEMP=UNSPEC(IOP(1));
   PUT FILE(SYSPRINT) EDIT ('PRODUCT= ',IOP(1),'BIT FORM= ',
        TEMP,'SIGN=',SR,'UQ= ',UQ,'OV=',OV) (SKIP,A,F(18,0),
       4(SKIP,A(10),B(65)) );
    PUT FILE(SYSPRINT) EDIT('END FO FX MPY') (SKIP,X(40),A);
    CALL TIMER;                          /*     END OF  FXMPY     */
    RETURN;
FLEND : CALL NORM ;
       SUBSTR(UQ,61,4)=MEPB;
MPYEND : FOP(1)=FOP(1)*FOP(2);  ON OVERFLOW GO TO NEXT;
   NEXT:  TEMP=UNSPEC( FOP(1));
    PUT FILE(SYSPRINT) EDIT('PRODUCT= ',FOP(1), 'BIT FORM= ',
     TEMP) (SKIP,A,E(16,6),SKIP,A,B(64));
    PUT FILE(SYSPRINT) EDIT ('SIGN= ',SR,'EXP= ',EUL,'UQ= ',UQ)
         (SKIP,A,B(1),SKIP,A,B(7),SKIP,A,B(64));
    PUT FILE(SYSPRINT) DATA (OV,UN,BOGUS,FA);
    SUBSTR(TEMP,1,8)=(8)'0'B;
    DIFF=(TEMP & ¬UQ)|( ¬TEMP & UQ) ;
    PUT FILE(SYSPRINT) EDIT ('DIFF= ',DIFF) (SKIP,A,B(64));
     PUT FILE(SYSPRINT) EDIT ('END OF FL MPY') (SKIP,X(40),A);
     CALL TIMER;
MEXTPRE : PROCEDURE;
     /* KEEP THE PRECISION OF MPY UP TO 2**-64        */
    DCL  PP(8) BIT(1),  (T1,T2,T3) BIT(1) PACKED;
     PP(8)=(8)'0'B;
    DO I=7 TO 1 BY -1;
        T1=SUBSTR(LS,I+57,1);
        T2=SUBSTR(LM,I+57,1);
        T3= ¬T2;
        PP(I)=(PP(I+1) & T3) |(T1 & T2)  ;
        END  ;
     PUT FILE(SYSPRINT) EDIT ('LS= ',SUBSTR(LS,58,7),
      'LM=>',SUBSTR(LM,58,7), 'PP= ', (PP(I) DO I=1 TO 8))
      (SKIP,A,B(8),SKIP,A,B(8),SKIP,A, 8 B(1)) ;
     DO I=1 TO 4  ;
        T1 =SUBSTR(LM,I+56,1) ;      /* T1=  LM (57 TO 60 FOR EACH I)
        T2=( PP(I) & ¬T1)|( ¬PP(I) & T1) ;  /* PP(I) .XOR. LM */
        SUBSTR(MEPB,I,1) = T2 ;
        END ;
     PUT FILE  (SYSPRINT)  EDIT  ('MEPB= ',MEPB)(SKIP,A,B(4));
     RETURN;  END;
      END MPY;
/*
```

```
//             1266,DCS,10.00,100,900),RTBUROVEC
//             EXEC PL1
//PL1.SYSPUNCH DD  UNIT=PUNCH,DSNAME=&PUNCH
//PL1.SYSIN    DD *
 DIV : PROCEDURE;
     /*  ............                              ........ */
     /*  . THIS ROUTINE INCLUDES  FLDIV  &  FX DIV      */
     /*  . 1.  THIS PART COMMON  TO  BOTH               */
     /*  .     GET  SR,NT, TEST  ZERO  DIVISOR,DIVIDENT */
     /*  . 2.  FOR FL                                   */
     /*  .     DIVIDEND IN  UQ0-7                       */
     /*  .     DIVISOR  IN  UH0-7                       */
     /*  .     NCC= NO. OF  DIVISION CYCLE =8           */
     /*  .     A).  SHFIT M(DIVISOR)  SO  DIVISOR IS    */
     /*  .          BETWEEN  1 & 1/2                    */
     /*  .     3).  CALL  DIVID                         */
     /*  .          TO PERFORM  MODEL DIVISION  NCC TIME*/
     /*  .     C).  CALL  ASSIMQD                       */
     /*  .          TO  ASSIMILATE  QUOTIENT            */
     /*  .     D).  SHIFT  QUOTIENT RIGHT AS DIVISOR    */
     /*  .          BEING SHIFT LEFT                    */
     /*  . 3.  FOR  FX                                  */
     /*  .     DIVIDEN  IN  UQ0-3                       */
     /*  .     DIVISOR  IN  UQ4-7                       */
     /*  .    A). TEST NFGATIVE  DIVISOR OR  DIVIDENT,  */
     /*  .        COMPL, THEN  SHIFT  DIVISOR IN UH0-3  */
     /*  .    B). SHIFT  DIVISOR  &  DIVIDEND  SO THAT  */
     /*  .        BIT1-4 OF UH BYTE 1  ¬= 0             */
     /*  .        GET  NCC                              */
     /*  .    C). CALL  DIVID                           */
     /*  .    D).  AT THIS  POINT                       */
     /*  .         REMAINDER IN  LM/LS, QUOTIENT IN  UH/UQ */
     /*  .         ASSIMILATE  REMAINDER                */
     /*  .    E).  ASSIMILATE  QUOTIENT                 */
     /*  .    F).  SHIFT REMAINDER RIGHT AS MANY TIMES AS */
     /*  .         DIVISOR HAS BEEN SHIFT  LEFT, PUT IT */
     /*  .         IN  UQ0-3                            */
     /*  .    G).  ASSIMILATED QUOTIENT IN  UQ4-7       */
     /*  ......                                         */
   DCL M BIT(64) PACKED EXT,  (LQ,UQ) BIT(65) PACKED EXT,  II EXT,
        (UH,LH,US,LS,UM,LM,X,Y,S,T,Z,N,G)  BIT(64) PACKED EXT,
     (P,B) BIT(64) PACKED EXT,
       (OV,UN,LSG,ID,GT,EQ,LT,FM,BOGUS) BIT(1) EXT,
       (EUU,EUM,EUL) BIT(7) EXT,
       (FA,FB) BIT(8) EXT,
       (SIGNA,SIGNB,POLIP,SDB,SR) BIT(1) EXT,
       V BIT(50) EXT , AEXPGT FIXED BIN ;
   DCL  IOP(2)  FIXED BIN(31) EXT,
        FOP(2)  FLOAT BIN(31) EXT,
        DOP(2)  DECIMAL FIXED(15) EXT;
    DCL NEGR EXT,  NCC EXT;
   DCL (M9,M10,M11,M12)  BIT(1),
       (D1,D2,D3,D4,D5,D6,D7,D8,D9,D10,D11) BIT(1),
       (DIVRL3,DIVRL2,DIVRL1) BIT(1),
       DIVSCC  FIXED BIN,  NEGO BIT(64),
       (PM,PS) BIT(6), AI BIT(7), BI BIT(7),
        QMM(8)  BIT(1)  PACKED,
```

```
      QSS(8) BIT(1)  PACKED,
      ICC  FIXED BIN ;
   DCL (ZEROP,TWOP,ZERON,TWON,ZERO,ONE,TWO) BIT(1),
      (A0,A1,A2,A3,A4,A5,A6) BIT(1);
   DCL    MARK CHARACTER(80) INITIAL ((80)'*');
   DCL   TEMP BIT(64),   BUFF BIT(6) INITIAL('100000'B);
   DCL (TEMP1,NFBM) BIT(1), TEMP2 BIT(2), TEMP4 BIT(4),
      TEMP6 BIT(6), TEMP8 BIT(8);
    DCL   NT BIT(2),   IV  BIT(4);
     /*  NFBM= NEGATE THE FIRST BIT TO MODEL DIVISION   */
      PUT FILE(SYSPRINT) EDIT ('START DIVISION') (SKIP,X(40),A);
      CALL TIMER;
   ON OVERFLOW  OV='1'B;   ON UNDERFLOW UN='1'B;
      ON ZERODIVIDE  BOGUS='1'B;
      NT=SUBSTR(V,8,2);
      IV=SUBSTR(V,4,4);
    SR= ( SIGNA & ¬SIGNB)|( ¬SIGNA & SIGNB);
      IF  NT='10'B  THEN GO TO  FLDIV;
      ELSE  GO  TO  FXDIV ;
FLDIV :  NCC = 8 ;
DZERO :     IF  UQ=(65)'0'B THEN DO ;
       EUL=(7)'0'B;  SR='0'B;
       GO TO DIVEND;  END;
    IF UH=(64)'0'B  THEN DO;
    OV='1'B;  SR=SIGNA;  UQ=(65)'1'B;  EUL=(7)'1'B;
    GO TO DIVEND;  END;
     /* .   THIS PART IS FOR  FL                          */
     /* .   SHIFT DIVISOR                                 */
    II=EUU-EUM   ; /*  II = DIFF OF EXPONENTS */
    DIVRL1,DIVRL2,DIVRL3='0'B;
    IF II> 63  THEN OV='1'B;    /* EUL>63 */
    IF II<-64  THEN UN='1'B;    /* EUL<-64 */
    CALL UHDM;
    CALL UQDUM;
    UQ=(65)'0'B;
    UH=(64)'0'B;
    IF SUBSTR(M,9,3)='000'B  THEN GO TO SHIFT3;  /* M(9-11)=0 */
    IF SUBSTR(M,9,2)='00'B   THEN GO TO SHIFT2;  /* M(9-10)=0 */
    IF SUBSTR(M,9,1)='0'B    THEN GO TO SHIFT1;  /* M(9)=0    */
    GO TO NOSHIFT;
 SHIFT3 : DIVRL3='1'B; PUT FILE(SYSPRINT) LIST ('DIVRL3') SKIP;
      CALL ML3Y3;
 NORDIR :  X=(64)'0'B;  S,G,N=X;
    CALL SDS(X,Y,S,G,N,T,Z);
    Y=(64)'0'B;  X=Z;  S=T;  GO TO STA;
  SHIFT2 : DIVRL2='1'B;  PUT FILE(SYSPRINT) LIST ('DIVRL2') SKIP;
      CALL ML2Y3;   GO TO NORDIR;
  SHIFT1 : DIVRL1='1'B;  PUT FILE(SYSPRINT) LIST ('DIVRL1') SKIP;
    CALL ML1Y4;
    X=(64)'0'B;  S,G,N=X;
 STA : CALL SDS(X,Y,S,G,N,T,Z);
    CALL Z4DLM;
       CALL LMDM;
  NOSHIFT : PUT FILE(SYSPRINT) EDIT ('M= ',M) (SKIP,A,B(56));
         CALL DIVID ;
                    N=(64)'1'B;  Y=(64)'0'B ;
    CALL UHDUS;
```

```
    CALL UQDUM;
       NEGO=(64)'1'B;
        US=( US & ¬NEGO)||( ¬US &NEGO);
          CALL  ASSIMQD;
/*   SINCE DIVISOR IS SCALED TO HAVE VALUE BETWEEN 1 AND 1/2
     THIS PART IS TO RESCALED IT  */
    PUT FILE(SYSPRINT) EDIT ('DIVRL123  ',DIVRL1,DIVRL2,DIVRL3,
       'LM BEFORE RECSCALE  ',LM) (SKIP,A, 3 B(1),SKIP,A,B(64));
    IF DIVRL1|DIVRL2|DIVRL3 THEN GO TO RESCALE;  ELSE GO TO NOSCALE;
RESCALE : CALL LMDM;
    IF DIVRL3 THEN CALL ML3Y3;
    ELSE IF DIVRL2 THEN CALL ML2Y3;
              ELSE    DO ;
                      CALL ML1Y4;
                      X,S,N,G=(64)'0'B;
                      GO TO SX1;  END;
        X,S,N,G=(64)'0'B;
        CALL SDS(X,Y,S,G,N,T,Z) ;  /* SDS3  */
        Y=(64)'0';  X=Z;  S=T;
SX1: CALL SDS(X,Y,S,G,N,T,Z);    /* SDS4 */
    CALL Z4DLM;
NOSCALE : PUT FILE(SYSPRINT) EDIT ('LM AFTER RESCALE= ',LM)
        (SKIP,A,B(64));
    CALL LMDUQ;
 NOR : CALL NORM ;
DIVEND :  BEGIN                ;
      CALL SETBOG;
    FOP(1)=FOP(1)/FOP(2);
    TEMP=UNSPEC(FOP(1));
    PUT FILE(SYSPRINT) EDIT ('QUOTIENT= ',FOP(1),'BIT FORM= ',TEMP)
         (SKIP,A,E(16,6),SKIP,A,B(65));
    SUBSTR (TEMP,1,8)=(8)'0'B ;
    TEMP=( TEMP & ¬UQ)||( ¬TEMP & UQ) ;
    PUT FILE(SYSPRINT) EDIT ('DIFF= ',TEMP) (SKIP,A,B(64));
    PUT FILE(SYSPRINT) EDIT ('SIGN= ',SR,'EXP= ',EUL,'UQ =',UQ,
         'FA= ', FA)     (SKIP,A,B(64));
     CALL TIMER;
     RETURN;
    END DIVEND;
     /* ....                                                  .... */
     /* . THIS PART IS FOR  FX                                     */
     /* . TEST NEGATIVE  DIVISOR OR  DIVIDEND,COMPLIMENT           */
FXDIV : NEGO=(64)'0'B;
    IF UH=(64)'0'B  THEN DO;
     OV='1'B;     UQ=(33)'0'B || (31)'1'B; SR=SIGNA;
     GO TO FXDIVEND;  END;
    /* FOR DIVID BY ZERO, RE=0,  Q=2**31-1    */
    IF SUBSTR(UQ,1,32)=(32)'0'B  THEN DO;
     UQ=(65)'0'B;  GO TO FXDIVEND;  END;
        SUBSTR(US,1,32)=SUBSTR(UQ,1,1) ||(31)'0'B;
        SUBSTR(US,33,32)=SUBSTR(UQ,33,1) || (31)'0'B;
        IF SUBSTR(UQ,1,1)='1'B THEN SUBSTR(NEGO,1,32)=(32)'1'B;
        IF SUBSTR(UQ,33,1)='1'B THEN SUBSTR(NEGO,33,32)=(32)'1'B;
        IF NEGO=(64)'0'B THEN GO TO OK;  /* BOTH POSITIVE */
NEGATE : G=(64)'1'B;  N,Y=(64)'0'B;
        US= ( US & ¬NEGO)||( ¬US & NEGO) ;
        CALL UQDUM;
```

```
        CALL UMDX1;
        CALL USDS1;
        CALL ASSIM;
        CALL Z4DLM;
        CALL LMDUQ3;
        CALL LMDUH7;
OK : SUBSTR (UQ,33,33)=(33)'0'B;   /* DIVIDEND IN UQ0-3 */
                                   /* DIVISOR IN UH0-3 */
        PUT FILE(SYSPRINT) EDIT ('UH= ',UH,'UQ= ',UQ) (SKIP,A,B(65));
        CALL TIMER;
   /*   SHIFT  DIVISOR  AND  DIVIDEND                              */
        NDRR8=0;
        IF SUBSTR(UQ,1,8) ¬= (8)'0'B  THEN GO TO SFTR8;
        IF SUBSTR(UH,1,8) =(8)'0'B  THEN GO TO NSFTR8;
SFTR8 :      CALL  UHDLH;          CALL UQDLQ;
             CALL  LHR8UH;         CALL LQR8UQ;
        NDRR8=1;
NSFTR8 : NDRL8=0 ;   NDDL8=0;
         NDRL4,NDRL1=0;
SFTL8  : IF  SUBSTR(UH,9,8)=(8)'0'B THEN DO;
         CALL  UHDLH;      CALL  LHL8UH;
         NDRL8=NDRL8+1;
         GO  TO  SFTL8 ;  END;
UQL8   : IF  SUBSTR(UQ,9,8)=(8)'0'B  THEN DO;
         CALL  UQDLQ;      CALL  LQL8UQ;
         NDDL8=NDDL8+1;
         GO TO  UQL8;      END;
UHL4   : IF  SUBSTR(UH,9,4)=(4)'0'B THEN DO;
         CALL  UQDLQ;      CALL UHDLH;
         CALL  LQL4UQ;     CALL LHL4UH;
         NDRL4 =1 ;
         END;
UHL1   : IF  SUBSTR(UH,9,1)='0'B THEN DO ;
         CALL  UHDLH;      CALL  UQDLQ;
         CALL  LHL1UH;     CALL  LQL1UQ;
         NDRL1=NDRL1+1;
         GO TO  UHL1;  END;
       NCC = NDRL8 - NDDL8 + 1;
       PUT FILE(SYSPRINT) EDIT ('AFTER SHIFT', 'UQ= ',UQ,
            'UH= ',UH) (SKIP,A, 2(SKIP,A,B(65)));
       PUT FILE(SYSPRINT) DATA (NDRL8,NDDL8,NDRL4,NDRL1,NCC) SKIP;
       PUT FILE(SYSPRINT) DATA (NDRR8) SKIP;
       GO TO DIVBEG;
    /*  ..................................................          */
    /*    THIS ENTRY IS FOR THE PART OF FX DIV WHICH IS             */
    /*        SHARED WITH CVD, FORM HERE TO THE END OF              */
    /*        OF ASSIMILATE REMAINDER,BEFORE ASSIMILATE             */
    /*        OF QUOTIENT                                           */
    /*  ..................................................          */
CVDDIV : ENTRY;
       IV=SUBSTR(V,4,4);
       CALL UHDUS;     GO TO SETUM;
         /*     IF CVD  THEN  UHDUS        , IF FXDIV  THEN UHDM */
DIVBEG : CALL UHDM;
   US=(64)'0'B;
SETUM : CALL UQDUM;
       UQ=(65)'0'B;
```

```
      UH=(64)'0'B;
      /* TEST FOR DIVIDEND < DIVISOR,                                  */
      /*     THEN SKIP DIVIDE PART, Q=0,RE=DIVIDEND                    */
      /*     BUT MUST BE RE-ASSIMILATE AND RE-SHIFT                    */
      IF NCC<=0  THEN GO TO ZEROQ;
      CALL DIVID ;
   /* . AL  HERE,  REMAINDER IN  LS/LM;                                */
   /* .            QUOTIENT  IN  UH/UQ;                                */
   /* . ASSIMILATE     REMAINDER FIRST;     INTO LM                    */
      CALL     LSDUS;
      CALL     LMDUM;
ZEROQ : NEGO=(64)'1'B;
   US= (US & ¬NEGO)|( ¬US & NEGO);
      Y=(64)'0'B;     N=(64)'1'B;
      CALL  ASSIMRE;
   /*  TEST NEGATIVE  REMAINDER,  ADD  M TO ASS. REMAINDER             */
    /*                            US & X UNCHANGED                    */
    /*  ALSO TEST SIGN OF DIVIDEND, SIGN OF RE = SIGNA                 */
    /*    FOR RE  +,   DIVIDEND  +,    OK                            */
    /*        RE  +,   DIVIDEND  -      RE= -RE                       */
    /*        RE  -,   DIVIDEND  +      RE = RE+ DIVISOR              */
    /*        RE  -,   DIVIDEND  -,     RE = -(RE+DIVIDOR)            */
      NEGR=0;
      IF SUBSTR(LM,1,1)='1'B  THEN DO ;
       /* NEG RE DUE TO DIVID ALGORITHM                                */
          NEGR=1;
      IF IV='0011'B  THEN Y='000000001010'B || (52)'0'B;
      ELSE CALL MDY1   ;
      /*  FOR CVD  DIVISOR=10 BUT M IS UESED FOR REMAINDER             */
      GO TO REASS2;    END;
    /* POSITIVE RE DUE TO DIVID ALGORITHM                             */
    /* CHANGE RE TO BE THE SAME SIGN AS DIVIDEND                      */
      IF  ¬SIGNA  THEN GO TO QASS ;
      Y=(64)'0'B;
REASS2 :  NEGO=(64)'1'B;
      N=(64)'1'B;
      IF SIGNA   THEN N=(64)'0'B;  /* NEG1 = ¬SIGNA  */
         CALL   ASSIMRE;
    /*    FOR CVD, RETURN TO CALLING PROGRAM AT HERE                   */
QASS:  IF IV='1110'B  THEN GO TO ASSQ;
      RETURN ;  /* END OF DIV FOR CVD                                 */
    /* THEN ASSIM  QUOTIENT;                                          */
   /*   MOVE  ASSIMLATED  REMAINDER TO UQ                             */
   /*  FOR NEGATIVE  REMAINDER , SUB 1 TO QUOTIENT                    */
   /*  BEFORE  ASSIMILATE                                            */
ASSQ:  CALL UHDUS;
      CALL UQDUM;
      CALL LMDUQ;
      NEGO=(64)'0'B;
      US=( US & ¬NEGO)|( ¬US & NEGO);
      M=(64)'0'B;
      IF  NEGR=1  THEN  SUBSTR(M,64,1)='1'B;
      CALL MDY1;
      IF  SR =1   THEN  N=(64)'1'B;
                  ELSE  N=(64)'0'B;
      CALL ASSIMOD;
   /*  .                                                             */
```

```
   /*  .   REMAINDER   HAS TO BE  SHIFT  RIGHT                    */
   /*  .   AS MANY TIMES  AS  DIVISOR HAS BEEN                    */
   /*  .   SHIFT LEFT
       IF NCC<1  THEN NDRL8=NDDL8;
RER8 : IF NDRL8 ¬=0  THEN DO;
       IF NDRR8=1  THEN DO ;
       NDRR8=0;   GO TO SKIPT;   END;
            CALL  UQDLQ;   CALL  LQR8UQ ;
       IF SIGNA  THEN SUBSTR(UQ,1,8)=(8)'1'B;
SKIPT :      NDRL8 =NDRL8 -1;
             GO  TO  RER8;       END;
       IF NDRR8=1 THEN DO;
       CALL UQDLQ;
       CALL LQL8UQ;
       END;
RER4:    IF  NDRL4  >0  THEN   DO;
             CALL  UQDLQ;        CALL  LQR4UQ;
       IF SIGNA  THEN  UQ=(4)'1'B || SUBSTR(UQ,5,61);
          /* UQ (BIT 1 - 4) = 1                                   */
            END;
RER1:     IF  NDRL1  >0   THEN   DO;
        CALL   UQDLQ;             CALL  LQR1UQ;
       IF SIGNA  THEN  UQ= '1'B  || SUBSTR(UQ,2,64);
          /* SET UQ (BIT 1 ) = 1                                  */
        NDRL1=NDRL1-1;
             GO TO  RER1;        END;
       CALL    LMUQ7;
FXDIVEND :   IOP(1)=IOP(1)/IOP(2) ;
      CALL SETBOG;
       TEMP=(64)'0'B;
       TEMP= UNSPEC (IOP(1));
       PUT FILE(SYSPRINT) EDIT  ('SIGN= ',SR, 'UQ= ',UQ,
            'BIT FORM= ', TEMP, 'FA= ', FA)
                 (SKIP,A(14), B(64)) ;
       PUT FILE(SYSPRINT) LIST ('FIX QUOTIENT= ', IOP(1)) SKIP;
    CALL TIMER;
     RETURN;
DIVID:  PROCEDURE;                                               */
     /*  PERFORM  NCC TIMES  DIVID CYCLE
    ICC=0 ;          /*  ICC= SYSLE COUNT     */
 BACK : PUT FILE(SYSPRINT) LIST (MARK) SKIP;
        CALL TIMER;
        IF ICC =NCC  THEN  GO TO     DIVIDEND;
 S1:DIVSCC=0;
    CALL DODMD;
    NFBM='0'B;
    CALL MODDIV;
    Y=(64)'0'B;
       IF IV='1110'B  THEN GO TO DIV1;
 CVD1 : IF QMM(1)  THEN CALL TENL7Y1;
        IF QMM(2)  THEN CALL TENL6Y1;
        GO TO TSS1;
 DIV1 : IF QMM(1)  THEN CALL ML7Y1;
    IF QMM(2) THEN  CALL ML6Y1;
 TSS1 : IF QSS(1)  THEN NEGO=(64)'1'B;
                   ELSE NEGO=(64)'0'B;    /* NEGO= QS1  */
    US =( US & ¬NEGO)|( ¬US & NEGO);
```

```
    CALL UMDX1;  CALL USDS1;  G=(64)'1'B;
    N=NEGO;     /* N=NEG1= QS1 */
    CALL SDS(X,Y,S,G,N,T,Z) ;
    TEMP2= SUBSTR(Z,1,2);
    IF TEMP2 ¬= '00'B  THEN NFBM='1'B;  ELSE NFBM='0'B;
    PUT FILE(SYSPRINT)  EDIT ('TEMP2= ',TEMP2) (A,B(2));
S2:DIVSCC=1;
    CALL D1DMD;
    CALL MODDIV;
    Y=(64)'0'B;
        IF IV='1110'B  THEN GO TO DIV2;
CVD2 : IF QMM(3)  THEN CALL TENL5Y2;
       IF QMM(4)  THEN CALL TENL4Y2;
       GO TO TSS2;
DIV2 : IF QMM(3)  THEN CALL ML5Y2;
    IF QMM(4)='1'B  THEN CALL ML4Y2;
TSS2 : IF QSS(3)  THEN      N=(64)'1'B;
                  ELSE N=(64)'0'B;    /*  N=NEG2= QS3 */
    S= ( T & ¬N)|( ¬T & N);    /* S= T.XOR.NEG2  */
    CALL SDS(Z,Y,S,G,N,T,Z);
    TEMP4= SUBSTR(Z,1,4);
    IF TEMP4 ¬=(4)'0'B  THEN NFBM='1'B;  ELSE NFBM='0'B;
    PUT FILE(SYSPRINT) EDIT ('TEMP4= ',TEMP4) (A,B(4));
S3:DIVSCC=2;
    CALL D2DMD;
    CALL MODDIV;
    Y=(64)'0'B;
        IF IV='1110'B  THEN GO TO DIV3;
CVD3 : IF QMM(5)  THEN CALL TENL3Y3;
       IF QMM(6)  THEN CALL TENL2Y3;
       GO TO TSS3;
DIV3 : IF QMM(5)  THEN CALL ML3Y3;
    IF QMM(6)  THEN CALL ML2Y3;
TSS3 : IF QSS(5)  THEN N=(64)'1'B;
                  ELSE N=(64)'0'B;    /*  N=NEG3= QS5 */
    S= ( T & ¬ N)|( ¬T & N);     /* S= T.XOR.NEG3  */
    CALL  SDS(Z,Y,S,G,N,T,Z);
    TEMP6=SUBSTR(Z,1,6);
    IF TEMP6 ¬=(6)'0'B  THEN NFBM='1'B;  ELSE NFBM='0'B;
    PUT FILE(SYSPRINT) EDIT ('TEMP6= ',TEMP6) (A,B(6));
S4:DIVSCC=3;
    CALL D3DMD;
    CALL MODDIV;
    Y=(64)'0'B;
        IF IV='1110'B  THEN GO TO DIV4;
CVD4 : IF QMM(7)  THEN CALL TENL1Y4;
       IF QMM(8)  THEN CALL TENDY4;
       GO TO TSS4;
DIV4 : IF QMM(7)  THEN CALL ML1Y4;
    IF QMM(8) THEN CALL MDY4;
TSS4 : IF QSS(7)  THEN N=(64)'1'B;
                  ELSE N=(64)'0'B;    /* N=NEG4= QS7 */
    S= ( T & ¬N)|( ¬T & N);       /* S= T.XOR.NEG4  */
    CALL SDS(Z,Y,S,G,N,T,Z);
     /*   FOR FX DIV, LAST CYCLE AND LAST SDS,SKIP                */
     /*            THIS NEGATE PART DUE TO OV IN THE SDS          */
        IF ICC ¬= (NCC-1)  THEN GO TO NEGA;
```

```
      IF NT ¬= '10'B  THEN GO TO NONEG;
NEGA : TEMP8=SUBSTR(Z,1,8);
   IF TEMP8 ¬=(8)'0'B THEN DO;
   TEMP='000000001'B || (55)'0'B;
   T= ( T & ¬TEMP) | ( ¬T & TEMP);     /* NEGATE 9TH BIT OF T  */
      PUT FILE (SYSPRINT) EDIT ('TEMP8= ',TEMP8)
       (A, B(8), X(5),A,B(1));
   PUT FILE(SYSPRINT) EDIT ('T AFTER NEGATE= ',T,
      'NEGATE SIGNAL=  ',TEMP) (SKIP,X(5),A,B(64));
      END;
/* FEED BACK LOWER REG. TO UPPER REG.*/
NONEG : CALL T4DLS;
   CALL Z4DLM;
   CALL LSL8US;          CALL LML8UM;
   CALL UHDLH;  CALL UQDLQ;
   CALL LHL8UH;          CALL LQL8UQ;
   CALL QBDUQH;
   PUT FILE(SYSPRINT) EDIT ('UH= ',UH,'UQ= ',UQ) (SKIP,A,B(65));
/* THEN REPEAT FOR NEXT CYCLE */
   ICC=ICC+1 ;
   GO TO BACK;
DIVIDEND : RETURN;
      END;                                                    */
    /*   ASSIMILATE REMAINDER OR QUOTIENT,                    */
    /*       PUT ASSIMILATED RESULT IN LM                     */
 ASSIMQD : PROCEDURE;
   PUT FILE(SYSPRINT) LIST('ASSIMILATE QUOTIENT') SKIP(2);
      GO TO ASS;
 ASSIMRE : ENTRY;
      PUT FILE(SYSPRINT) LIST ('ASSIMILATE REMAINDER') SKIP(2);
 ASS :  CALL USDS1 ;
      CALL  UMDX1 ;
      CALL ASSIM ;
      CALL Z4DLM ;
      END  ASSIMQD;
 MODDIV : PROCEDURE ;
   DCL (TB,TPM,TPS) BIT(1),  PMT BIT(7);
   IF NFBM  THEN DO;
      TEMP1=SUBSTR(PS,1,1);
   PS = ( PS & ¬BUFF) | ( ¬PS & BUFF);    /* NEGATE 1ST BIT FO PS */
      PUT FILE(SYSPRINT) EDIT ('NFBM= ',NFBM)
     (A,B(1), X(5), A, B(1));
   PUT FILE(SYSPRINT) EDIT ('PS AFTER NEGATE= ',PS,
      'NEGATE SIGNAL=  ',BUFF) (SKIP,X(5),A,B(6));
      END;
   SUBSTR(BI,7,1)='0'B;
   DO I=6 TO 1 BY -1;
    TB=SUBSTR(BI,I+1,1);
    TPM=SUBSTR(PM,I  ,1);  TPS=SUBSTR(PS,I  ,1);
   SUBSTR(BI,I,1)=(TPM & TPS)|( ¬TPM & TB);
   END;
   PMT='0'B || PM;  PUT FILE(SYSPRINT) DATA (PMT);
   AI=(PMT & ¬BI)|( ¬PMT &  BI);
   PUT FILE (SYSPRINT) EDIT ('MODEL DIVISION','CYCLE=',ICC,
     'SDS ',DIVSCC,'PM=',      PM,'PS=', PS,'B=',BI,'A=',AI)
     (SKIP,A,SKIP,A,F(1),X(5),A,F(1),4(SKIP,A,B(7)));
   IF (DIVSCC=0 & ICC=0)  THEN DO;
```

```
SELECT : BEGIN;                    /* SELECT DIVISOR INTERVAL */
    IF IV='0011'B  THEN DO;        /*    FOR  CVD    */
      D3,D7,D11='1'B; GO TO SLEND;  END;
       M9=SUBSTR(M,9,1) ;
       M10=SUBSTR(M,10,1);
       M11=SUBSTR(M,11,1);
       M12=SUBSTR(M,12,1);
       D1= ¬M10 &¬M11 & ¬M12;      /* 1/2 */
       D2= ¬M10 & ¬M11 & M12;      /* 9/16 */
       D3= ¬M10 & M11 & ¬M12;     /* 5/8  */
       D4= ¬M10 &  M11 &  M12;     /* 11/16  */
       D5=  M10 & ¬M11 ;           /* 3/4 OR 13/16 */
       D6=  M10 &  M11 ;           /* 7/8 OR 15/16 */
       D7= D1|D2|D3;
       D8= D4|D5|D6    ;
       D9= D4|D5       ;
       D10=D5|D6       ;
    D11 = D7|D4;
SLEND : END SELECT;
   PUT FILE(SYSPRINT) EDIT ('D1 TO D11',D1,D2,D3,D4,D5,D6,D7,D8,
     D9,D10,D11) (SKIP,A,X(5), 11(B(1),X(1)) );
   END;   /* DO LOOP */
/*      THIS PART IS USED TO SELECT
       QUOTIENT                  */
QUOTIENT : BEGIN;
       A0=SUBSTR(AI,1,1);
       A1=SUBSTR(AI,2,1);
   A2=SUBSTR(AI,3,1);
       A3=SUBSTR(AI,4,1);
       A4=SUBSTR(AI,5,1);
       A5=SUBSTR(AI,6,1);
       A6=SUBSTR(AI,7,1);
   ZEROP= ( ¬A0 & ¬A1 & ¬A2 & ¬A3 & ¬A4)
         |( ¬A0 & ¬A1 & ¬A2 & ¬A3 & A4 & ¬A5 & ¬D1)
         |( ¬A0 & ¬A1 & ¬A2 & ¬A3 & D10) ;
   TWOP = ( ¬A0              & A3 & A4 & ¬A5 & A6 & D1)
         |( ¬A0             & A3 & A4 & A5 &D1)
         |( ¬A0              & A3 & A4 & A5 & A6 & D2)
         |( ¬A0 & A2 & A3 & D8)
         |( ¬A0        & A2 & ¬A3 & A4 & D9)
         |( ¬A0        & A2 & ¬A3 & ¬A4 & A5 & D4)
         |( ¬A0        & A2 & ¬A3 & A4 & A5 & A6 & D6)
         |( ¬A0 & A1) | ( ¬A0 & A2 & D7);
   ZERON= ( A0 & A1 & A2 & A3 & A4)
         |( A0 & A1 & A2 & A3 & ¬A4 & A5 & D10);
   TWON=  ( A0            & ¬A3 & ¬A4 & D1)
         |( A0            & ¬A3 & ¬A4 & ¬A5 & D2)
         |( A0            & ¬A3 & ¬A4 & ¬A5 & ¬A6 & D3)
         |( A0       & ¬A2 & A3 & A4 & ¬A5 & ¬A6  & D5)
         |( A0       & ¬A2 & A3 & ¬A4 & ¬A5 & D6)
         |( A0 & ¬A2 & D11 )
         |( A0       & ¬A2 & A3 & ¬A4 & D5)
         |( A0       & ¬A2 & ¬A3 )
         |( A0 & ¬A1) ;
   ZERO=ZERON|ZEROP;
   TWO= TWON|TWOP;
   ONE = ¬ZERO & ¬TWO;
```

```
   END QUOTIENT ;
   A0=A0 & ¬ZERO;    /* ALWAYS TO SUBTRACT 0 FROM PARTIAL REMAINDER*/
   IF DIVSCC=0 THEN DO ;
       CALL LDQMS12;GO TO OUT;  END;
   IE DIVSCC=1 THEN DO ;
       CALL LDQMS34;GO TO OUT;  END;
   IF DIVSCC=2 THEN DO;
       CALL LDQMS56;  GO TO OUT; END;
   CALL   LDQMS78;
OUT: PUT FILE(SYSPRINT) DATA (ZEROP,ZERON,TWOP,TWON,TWO,ZERO,ONE);
   RETURN;
   END  MODDIV;
DODMD: PROCEDURE;
   PM= SUBSTR(UM,1,6);
   PS=SUBSTR(US,1,6);
   RETURN; END;
D1DMD: PROCEDURE;
   PM=SUBSTR(Z,3,6);
   PS=SUBSTR(T,3,6);
   RETURN;  END;
D2DMD : PROCEDURE      ;
   PM= SUBSTR(Z,5,6);
   PS= SUBSTR(T,5,6);
   RETURN;  END;
D3DMD : PROCEDURE;
   PM= SUBSTR (Z,7,6);
   PS =SUBSTR (T,7,6);
   RETURN; END;
LDQMS12 : PROCEDURE ;
   QMM(1)=TWO;     QSS(1)=A0;
   QMM(2)=ONE;     QSS(2)=A0;
   PUT FILE(SYSPRINT) DATA (QMM(1),QMM(2),QSS(1),QSS(2));
   RETURN; END;
LDQMS34: PROCEDURE;
   QMM(3)=TWO;    QSS(3)=A0;
   QMM(4)=ONE;    QSS(4)=A0;
   PUT FILE(SYSPRINT) DATA (QMM(3),QMM(4),QSS(3),QSS(4));
   RETURN; END;
LDQMS56 : PROCEDURE;
   QMM(5)=TWO;    QSS(5)=A0;
   QMM(6)=ONE;    QSS(6)=A0;
   PUT FILE(SYSPRINT) DATA (QMM(5),QMM(6),QSS(5),QSS(6));
   RETURN;  END;
LDQMS78:  PROCEDURE ;
   QMM(7)=TWO;    QSS(7)=A0;
   QMM(8)=ONE;    QSS(8)=A0;
   PUT FILE(SYSPRINT) DATA (QMM(7),QMM(8),QSS(7),QSS(8));
   RETURN; END;
QBDUQH : PROCEDURE;
   DO  I=1 TO 8;
   SUBSTR(UQ,I+56,1)=QMM(I);
   SUBSTR(UH,I+56,1)=QSS(I);
   END;
   RETURN;  END;
LMDUQ3 : PROCEDURE;
        SUBSTR(UQ,1,32)=SUBSTR(LM,1,32);
        END;
```

```
 LMDUH7 : PROCEDURE;
        SUBSTR(UH,1,32)=SUBSTR(LM,33,32);
        END;
 LQL1UQ :  PROCEDURE;
        UQ=(65)'0'B;
        SUBSTR(UQ,1,64)=SUBSTR(LQ,2,64) ;
        END ;
LQR1UQ  :  PROCEDURE;
        UQ=(65)'0'B ;
        SUBSTR(UQ,2,64)=SUBSTR(LQ,1,64);
        END ;
 LHL1UH  :   PROCEDURE      ;
        UH=(64)'0'B;
        SUBSTR(UH,1,63)=SUBSTR(LH,2,63) ;
        END       ;
LMUQ7 : PROCEDURE ;
        SUBSTR(UQ,33,32)=SUBSTR(LM,33,32);
        END;
 TENL7Y1 : PROCEDURE;       /*  Y1(2&4)=1                          */
        Y='0101'B || (60)'0'B;
        END;
 TENL6Y1 : PROCEDURE;  /*    Y1(3&5)=1                             */
        Y='00101'B || (59)'0'B;
        END;
 TENDY1 : PROCEDURE;          /*    Y1(9 & 11)=1                   */
        Y='00000000101'B || (53)'0'B;
        END;
 TENL5Y2 : PROCEDURE ;         /*    Y2( 4 & 6)=1                  */
        Y='00010100'B  || (56)'0'B;
        END;
 TENL4Y2 : PROCEDURE;        /*    Y2( 5 & 7)=1                    */
        Y='00001010'B || (56)'0'B;
        END;
 TENL3Y3 : PROCEDURE;      /* Y3(6 & 8)=1                          */
        Y='00000101'B ||(56)'0'B;
        END;
 TENL2Y3 : PROCEDURE ;      /*  Y3(7 & 9)=1                        */
        Y='000000101'B ||(55)'0'B;
        END;
 TENL1Y4 : PROCEDURE;         /*    Y4(8 10) =1                   */
        Y='0000000101'B || (54)'0'B;
        END;
 TENDY4 : PROCEDURE;       /*  Y4(9 &11)=1                        */
        Y='00000000101'B ||(53)'0'B;
        END;
    END  DIV ;
```

```
//              EXEC LKGOPL1
//              1235,DCS,03.00,100,900),'D. E. ATKINS',MSGLEVEL=1
//              EXEC  PL1
//PL1.SYSPUNCH DD  UNIT=PUNCH,DSNAME=&PUNCH
//PL1.SYSIN    DD *
   CONVS : PROCEDURE;
   /*                                                              */
   /*  THIS PROCEDURE INCLUDES ALL CONVERTING TYPE INSTRUCTION  */
   /* IT INCLUDES  : DVB (DEC TO BIN)                             */
    /*               FXTO FL : BIN TO FL                            */
   /*               FXCVFL  FX(BYTE 1 TO 4 OF UQ) TO FL          */
   /*                                                             */
   /*
      DCL (M,UH,LH,US,LS,UM,LM,X,Y,S,T,Z,N,G) BIT(64)  PACKED EXT,
          (LQ,UQ) BIT(65) PACKED EXT,  II EXT,
          (P,B) BIT(64) PACKED EXT,
          (OV,UN,LSG,ID,GT,EQ,LT,FM,BOGUS) BIT(1) EXT,
          (EUU,EUM,EUL) BIT(7) EXT,
          (FA,FB) BIT(8) EXT,  (SIGNA,SIGNB,POLIP) BIT(1) EXT,
          V BIT(50) EXT,(SDB,SR) BIT(1)EXT, AEXPGT FIXED BIN EXT;
      DCL IOP(2) FIXED  BIN(31) EXT,
          FOP(2) FLOAT  BIN(53) EXT,
          DOP(2) DECIMAL FIXED (15) EXT,
          OPF(2) BIT(8), FLAG(4) BIT(4);
       DCL NEGR EXT,  NCC EXT;
      DCL TEMP BIT(64), MARK CHAR(80) INITIAL ((80)'*');
       DCL NT BIT (2);
       DCL NEGO BIT(64);
   /*  ..................................................................
       /* .  THIS ROUTINE IS USED TO CONVERT EITHER DEC TI FX    */
                                                  OR FL TO FX    */
       /* .                                                      */
       /* ..................................................... */
   /*  ..................................................................
   CVL :  ENTRY ;
       ON OVERFLOW    BOGUS='1'B;
          NT=SUBSTR(V,8,2);
          IF NT='11'B THEN GO TO DECCVFX;   /* DEC TO FX  */
          IF NT='10'B  THEN  GO TO  FLCVFX;  /*  FL  TO  FX    */
          ELSE GO TO  OUT;
   DECCVFX: PUT FILE(SYSPRINT) EDIT ('START DEC TO FX') (SKIP,X(40),A
          CALL TIMER;
              CALL DVB;
   /* DEC  TO FX
            /*  FOR CVL (DEC TO FX), BIN DIG. IS RIGHT ADJUST IN UQ *
   /*
   DECTOFX : IOP(1)=DOP(1);     /* 360 DEC TO FX  */
      PUT FILE(SYSPRINT) LIST ('DECIMAL TO FIXED') SKIP(2);
      IF SUBSTR(UQ,1,32) ¬=(32)'0'B THEN OV='1'B;
   TESTNEG:IF SIGNA= '0'B THEN GO TO FXEND;
      PUT FILE(SYSPRINT) LIST ('COMPLIMENT NEGATIVE NO.') SKIP(2);
          US=(64)'0'B;  NEGO=(64)'0'B;
       CALL COMPL;
    FXEND :  TEMP=UNSPEC(IOP(1));
        IF OV THEN CALL SETBOG;
       SR=SIGNA;      /* SIGN OF RESULT  */
           PUT FILE(SYSPRINT) EDIT ('SIGN= ',SR,'UQ= ',
            UQ,'ANS=',TEMP,'FA=',FA) (SKIP,A,B(64));
          PUT FILE(SYSPRINT) EDIT ('END OF CVL       ') (SKIP,X(40),A)
```

```
        CALL TIMER;
        RETURN;
    /* .
    /*   THIS PART  CONVERT  FL TO FX                              */
    /*   FL IN  UQ  1-7 ,    EXPONENT IN EUM                       */
    /* .                                                           */
FLCVFX : PUT FILE(SYSPRINT) EDIT ('START FL TO FX') (SKIP,X(40),A);
        CALL TIMER;
        IOP(1)=FOP(1);
            II = EUM - 64 ;
        IF  II >0  THEN  GO  TO  NFRACT;
ZOUT:   UQ=(65)'0'B;
        GO TO FXEND;
NFRACT :  IF  II > 21  THEN DO;
             OV='1'B;
             GO TO  ZOUT;          END;
          IF  II >8  THEN  OV='1'B ;
          IEUL= 14 -II ;
          IEUU= IEUL ;
          IF   IEUU <0   THEN  GO TO  SFTL8;
          ELSE   GO TO     SFTR8 ;
SFTR8 :  IF IEUU  >=2  THEN  DO ;
             CALL   UQDLQ ;
             CALL  LQR8UQ;
             IEUU=IEUU- 2;
             GO TO  SFTR8 ;       END;
        IF IEUU=1  THEN DO ;
             CALL  UQDLQ;
             CALL  LQR4UQ;
             END ;
        GO TO  GETFX;
SFTL8 :IF  IEUU <=  -2   THEN DO ;
             CALL  UQDLQ;
             CALL  LQL8UQ;
             IEUU=IEUU+2 ;
             GO  TO  SFTL8;       END;
        IF IEUU = -1  THEN  DO;
             CALL  UQDLQ;
             CALL LQL4UQ;         END;
GETFX:   IF SUBSTR(UQ,33,1)='1'B THEN      DO;
             OV='1'B      ;
             END ;
             GO TO TESTNEG;
OUT :   CALL TIMER;
        RETURN;
/*  ..................................................................  */
/*  ..................................................................  */
CVF : ENTRY;
    /* ..................................................  */
    /* ..................................................  */
    /* .  THIS ROUTINE IS USED TO CONVERT                  */
    /* .   1.  DEC TO FL BY CALLING DVB, BINARY DIGITS ARE */
    /* .       RIGHT ADJUST IN UQ, THEN SET EUL=14,        */
    /* .      USING FX TO FL                               */
    /* .   2.  FX TO FL SET EUL=8                          */
    /* ..................................................  */
    /* ..................................................  */
```

```
      NT = SUBSTR(V,8,2);
      IF NT='11'B THEN GO TO DECCVFL;
        ELSE GO TO FXCVFL;
DECCVFL : PUT FILE(SYSPRINT) EDIT ('START DEC TO FL ')
        (SKIP,X(40),A);
      CALL TIMER;
      CALL DVB;
  /* FOR CVF(DEC TO FL) , AFTER DVB(DEC TO BIN) THEN TO FL  */
DECTOFL : FOP(1)=DOP(1);   /* CONVERT DEC TO FL BY 360 */
/*  FX TO FL (AFTER DVB)        */
/*  FX IS RIGHT  ADJUST  IN UQ */
   II=14;            /* EUL=14  MAX EXP, THEN NORMOLIZATION */
FXTOFL : CALL NORM;
   SR=SIGNA;      /* SIGN OF RESULT */
      PUT FILE(SYSPRINT) LIST ('CVF OUTPUT');
CVFEND : PUT FILE(SYSPRINT)
           DATA (SR,EUL,UQ,FA) SKIP(2) ;
   TEMP=UNSPEC(FOP(1)) ;
   PUT FILE(SYSPRINT) EDIT ('360 OUTPUT', 'FL =',FOP(1),
      'BIT= ',TEMP) (SKIP,A,SKIP,A,E(16,6),SKIP,A,B(64));
      CALL TIMER;
   RETURN;                                                    */
/*  ..........                                                */
/*  ..........                                                */
/*  THES PART     IS USED TO                                  */
/* 2. CONVERT  FX TO FL  BY FXTOFL                           */
/*                                                           */
/*  FIX NO. TO BE CONVERT IS IN BYTE 0-3 IN UQ          */
/*   FIRST SHIFT TI TO BYTE 1-4 IN UQ,  THEN TEST FOR NEGATIVE */
/*  ..........                                                */
/*  ..........                                                */
FXCVFL : II=8;     /* EUL=8  MAX EXP  */
   CALL UODLQ;
   CALL LOR8UQ;
   SR=SIGNA;
        FOP(1)=IOP(1);
   IF SUBSTR(UQ,9,1)='0'B THEN GO TO FXTOFL;  /* POSITIVE */
COMPLE: SIGNA='1'B;
      NEGO=(64)'1'B;
      US='000000001'B || (55)'0'B;  /* 9TH BIT IS SIGN  */
       CALL COMPL;
       GO TO FXTOFL;                                          */
/*  ..........                                                */
/*  ..........                                                */
DVB : PROCEDURE;                                             */
/*  ..........                                                */
/*  THIS ROUTINE IS USED TO  CONVERT  DEC TO BIN , RESULT  IN UQ, */
/*                                                            */
/*  MAY  BE CALL BY CVL OR CVF
/*  TEST  ZERO
              */                                             */
/*  ..........                                                */
    /* ..........                                             */
    /* ..........                                             */
      PUT FILE(SYSPRINT) LIST ('START DEC TO BINARY CONVERSION')
        SKIP;
   IF UQ=(65)'0'B THEN GO TO DVBEND;
   IF SUBSTR(UQ,9,52)=(60)'0'B THEN GO TO  DVBEND;
```

```
   ICC=14;          /* TOTAL 14 DEC DIGIT */
LSHIFT: IF SUBSTR(UQ,5,8)=(8)'0'B THEN DO;
            CALL UQDLQ;
            CALL LQL8UQ;
            ICC=ICC-2;
            GO TO  LSHIFT;    END;
        ELSE IF SUBSTR(UQ,5,4)='0000'B THEN DO;
            CALL UQDLQ;
            CALL LQL4UQ;
            ICC=ICC-1;  END;
/* M(61-64)=UQ(5-8) */
   SUBSTR(M,61,4)=SUBSTR(UQ,5,4);
   PUT FILE(SYSPRINT) EDIT ('UQ= ',UQ,'M=  ',M) (SKIP,A,B(64));
/*                  */
/*  LM=10*M + UM                          */
/*  PARTIAL RESULT IN LM  THEN TO M */
/*  NEXT DIGIT GET FROM  UQ TO UM */
/*                  */
/* UM(61-64)=UQ(9-12) */
/*                  */
NXTDIG : UM=(64)'0'B;            US=UM;
         SUBSTR(UM,61,4)=SUBSTR(UQ,9,4);
   PUT FILE(SYSPRINT) EDIT ('DIGIT= ',ICC,'UM= ',UM)
       (SKIP,X(70),A,F(2,0),SKIP,A,B(64));
SDS1:  N,Y,S,G=(64)'0'B;
       G=(64)'1'B;
       CALL USDS1;
       CALL UMDX1;
       CALL SDS(X,Y,S,G,N,T,Z);
       N=(64)'1'B;  /* N2= 1 */
SDS2:  CALL SDS(Z,Y,T,G,N,T,Z);
       N=(64)'0'B;    /* N3= 0 */
SDS3:  CALL ML3Y3;
       CALL SDS(Z,Y,T,G,N,T,Z);
       N=(64)'1'B;     /* N4= 1 */
SDS4:  CALL ML1Y4;
       CALL SDS(Z,Y,T,G,N,T,Z);
   CALL T4DLS;
   CALL Z4DLM;
   CALL LMDUM;
   CALL LSDUS;
/* ASS1M */
   PUT FILE(SYSPRINT) LIST ('BEGIN ASSIM') SKIP(2);
   CALL UMDX1; CALL USDS1;  Y=(64)'0'B ;  N=Y;
   CALL ASSIM;
   CALL Z4DLM;       /* LM= ASSIMILATED OUTPUT */
   ICC=ICC-1;
/* END OF ONE DEGIT  */
/* TEST FOR ANY MORE DIGIT */
   IF ICC=0  THEN  DO ;
       CALL  LMDUQ;      /* FINAL BINARY IN UQ  */
       GO TO DVBEND;
       END ;
MORDIG:CALL UQDLQ;
       CALL LQL4UQ;
       CALL LMDM;
      PUT FILE(SYSPRINT) LIST(MARK) SKIP;
```

```
      PUT FILE(SYSPRINT) EDIT('UQ= ',UQ,'M= ',M) (SKIP,A,B(64));
      GO TO NXTDIG;
DVBEND:  PUT FILE(SYSPRINT) LIST ('END OF CONVERT DECIMAL TO BINARY,
   BINARY DEGITS ARE REGHT ADJUST IN UQ') SKIP(2);
      CALL TIMER;
      RETURN;
    END DVB;
CVD : ENTRY;
   /* .  THIS ROUTINE IS USED TO CONERT  FL/FX TO DEC          */
   /* .  1. FOR FX    OPRAND  IS RIGHT ADJUST IN  UQ4-M        */
   /*                 IF NECESSARY, COMPL. NEG. NO.            */
   /*     2. FOR FL   OPRAND  IS IN  UQ0-7                     */
   /*                 IF IEUU=14  NO SHIFT                     */
   /*                 IF IEUU BETWEEN 0,14  SHIFT RIGHT        */
   /*                 IF  28<IEUU <14     SHIFT LEFT           */
   /*                 ELSE  UQ=0                               */
   /*     3. THEN USE FIX DIVID BY 10, QUOTIENT BECOME         */
   /*        NEW DIVIDEND UNTIL QUOTIENT IS 0                  */
   /*        IF  EUU=14  BOGUS=LOW ORDER 14 DIGIT              */
   /*            EUU>14  BOGUS UNDEFIEND DUE TO THE L-SHIFT    */
      NT=SUBSTR(V,8,2);
      SR =SIGNA;      /* SIGN OF RESULT */
      SIGNA='0'B;     /* CLEAR SIGNA FOR FX DIV */
      ON OVERFLOW  BOGUS='1'B;
      IF  NT ¬='10'B THEN GO TO  FXCVD;
FLCVD :II= EUM - 64;
      CALL TIMER;
      PUT FILE(SYSPRINT) EDIT ('START CONVERT FL TO DEC')
          (SKIP,X(40),A);
      DOP(1)=FOP(1);
      IF  II >0  THEN GO TO NFRAC ;
ZDEC : UQ=(65)'0'B;  GO TO SETSIGN;
NFRAC  :  IF II >28  THEN DO;
          OV='1'B;
          GO TO ZDEC;  END;
       II=14-II;     IF II=0   THEN GO TO GETDEC;  /*EXP=14 */
       IF II <0 THEN GO TO SHTL8;
       ELSE GO TO  SHTR8;                          /* 0<EXP<14   */
SHTR8 : IF II >=2 THEN DO;
          CALL  UQDLQ;
          CALL  LQR8UQ;
          II =II-2;
          GO TO SHTR8;  END;
        IF II=1  THEN DO;
          CALL  UQDLQ;
          CALL  LQR4UQ;   END;
        GO TO  GETDEC;                             /*  14<EXP<28  */
SHTL8:  IF  II <= -2  THEN DO;
            CALL UQDLQ;
            CALL LQL8UQ;
            II=II+2;
            GO TO SHTL8;   END;
         IF  II=-1 THEN DO;
            CALL UQDLQ;
            CALL LQL4UQ;   END     ;
        GO TO GETDEC;
FXCVD :CALL TIMER;
```

```
        PUT FILE(SYSPRINT) EDIT ('START CONVERT FX TO DEC')
            (SKIP,X(40),A);
        DOP(1)=IOP(1);
          IF SUBSTR(UQ,33,1)='0'B THEN GO TO GETDEC;  /* POS. FX */
          SUBSTR(US,33,32)='1'B ||(31)'0'B;      /* US(33)=1  */
        NEGO=(32)'0'B || (32)'1'B;
        CALL COMPL;
GETDEC : PUT FILE(SYSPRINT) EDIT ('UQ TO BE CONVERT= ',UQ)
             (SKIP,A,A(64));
        SUBSTR(UQ,65,1)='0'B;
        ND=0;
NXTDIV : PUT FILE(SYSPRINT) LIST(MARK) SKIP;
        PUT FILE(SYSPRINT) EDIT ('DIVIDEND=(UQ)=',UQ, 'UH= ',UH)
            (SKIP,A(18),B(65));
        IF UQ=(65)'0'B THEN GO  TO          GETDIG;
        ND=ND+1;
        PUT FILE(SYSPRINT) DATA(ND) SKIP;
        IF ND=15  THEN DO;
          OV='1'B;   GO TO SETSIGN;     END;
        CALL MDY4;
        X,S,G=(64)'0'B;   N=(64)'1'B;
        CALL SDS(X,Y,S,G,N,T,Z);                    /* SDS2 */
        CALL Z4DLM;     CALL LMR4M;
        NDDL8=0;
TUQ1 :  IF  SUBSTR(UQ,9,8)=(8)'0'B  THEN DO;
        CALL UQDLQ;     CALL UHDLH;
        CALL LQL8UQ;    CALL LHL8UH;
        NDDL8 = NDDL8 +1;
        GO TO TUQ1;     END;
LSHF4:  CALL UQDLQ;
        CALL LQL4UQ;
        CALL UHDLH;
        CALL LHL4UH;
        NCC= 7-NDDL8;
        PUT FILE(SYSPRINT) EDIT ('BEFORE DIVISION', 'ND= ', ND,
          'UH= ',UH,'UQ= ',UQ,'NO. OF DIVID CYCLE= ',NCC)
           (SKIP,A,SKIP,A,F(2,0),2(SKIP,A,B(65)),SKIP,A,F(2,0));
   /*   PERFORM  NCC  CYCLE  OF MODIFIED  FIXED DIVISION      */
   /*   ASSIMILATE REMAINDER,CORRECT ASNEEDED                 */
   /*   AFTER ASS. REMAINDER SHOULD BE  IM  LM (BIT 9-12)     */
        CALL CVDDIV;
        SUBSTR(M,9,4)=SUBSTR(LM,9,4);     /*  M(9-12)=LM(9-12) */
        PUT FILE(SYSPRINT) DATA (M) SKIP;
   /*   TEST THE ASSIMILATION OF REMAINDER, IF NEGATIVE       */
   /*   REMAINDER(ADD DIVISOR),  SUBTRACT 1 FROM QUOTIENT,    */
   /*   BUT DO NO ASS. QUOTIENT AT THIS POINT                 */
   /*                                                         */
        IF  NEGR=0  THEN GO TO NOSUB;
QSUB1 :CALL UHDUS;
        CALL  UQDUM;
        G=(64)'1'B;
        NEGO,N=(64)'0'B;
        Y=(63)'0'B || '1'B;
        CALL USDS1;
        CALL UMDX1;
        PUT FILE(SYSPRINT) LIST ('SUBTRACT QUOTIENT BY 1 ') SKIP;
        CALL SDS(X,Y,S,G,N,T,Z);                    /* SDS1 */
```

```
        Y=(64)'0'B;                                          /* SDS2 */
        CALL SDS(Z,Y,T,G,N,T,Z);                             /* SDS3 */
        CALL SDS(Z,Y,T,G,N,T,Z);                             /* SDS4 */
        CALL SDS(Z,Y,T,G,N,T,Z);
        CALL Z4DLM;
        CALL T4DLS;
        CALL LSDUH;
        CALL LMDUQ;
NOSUB  :CALL UHDUS;     CALL UQDUM;
        GO TO     NXTDIV;
GETDIG : CALL MDY4;
          X,S,G,N=(64)'0'B;
        PUT FILE(SYSPRINT) LIST ('MOVE M_REG TO UQ_REG') SKIP;
          CALL SDS(X,Y,S,G,N,T,Z);                           /* SDS4  */
          CALL Z4DLM    ;
          UQ=(65)'0'B;
          CALL LMDUQ;                                                */
    /* RIGHT ADJUST M REGISTER
RADJ  : IF ND<=12  THEN DO;
          CALL UQDLQ;   CALL LQR8UQ;
          ND=ND+2;  GO TO RADJ;  END;
        IF ND=13  THEN DO;
          CALL UQDLQ;   CALL LQR4UQ;   END;
SETSIGN: IF  SR THEN  SUBSTR(UQ,1,8)='00001011'B;  /* NEG   */
          ELSE  SUBSTR(UQ,1,8)='00001010'B;        /* POS   */
        IF OV THEN CALL SETBOG;
        PUT FILE(SYSPRINT) LIST ('CONVERTED  NO = ', DOP(1)) SKIP;
        PUT FILE(SYSPRINT) EDIT (' UQ= ',UQ,'OV= ',OV,'FA= ',FA)
             (SKIP,A,B(64));
        PUT FILE(SYSPRINT) EDIT ('END OF  CVD') (SKIP,X(40),A);
        CALL TIMER;
        RETURN;
LMR4M  : PROCEDURE;
        M='0000'B || SUBSTR(LM,1,60);
        RETURN;   END;
COMPL  : PROCEDURE;
          N,Y=(64)'0'B;  G=(64)'1'B;
        US= ( US & ¬NEGO)|( ¬US & NEGO);
         CALL USDS1;
         CALL UQDUM ;
         CALL UMDX1;
         CALL ASSIM;
         CALL Z4DLM;
         CALL LMDUQ;
         RETURN;  END COMPL;
  END CONVS;
/*
```

rm AEC–427
(6/68)
AECM 3201

U. S. ATOMIC ENERGY COMMISSION

UNIVERSITY–TYPE CONTRACTOR'S RECOMMENDATION FOR DISPOSITION OF SCIENTIFIC AND TECHNICAL DOCUMENT

(See Instructions on Reverse Side)

AEC REPORT NO. 418 COO-2118-0002	2. TITLE ARITHMETIC UNIT OF ILLIAC III: SIMULATION AND LOGICAL DESIGN-PART II

TYPE OF DOCUMENT (Check one):

- [X] a. Scientific and technical report
- [ ] b. Conference paper not to be published in a journal:
  - Title of conference _____
  - Date of conference _____
  - Exact location of conference _____
  - Sponsoring organization _____
- [ ] c. Other (Specify) _____

RECOMMENDED ANNOUNCEMENT AND DISTRIBUTION (Check one):

- [X] a. AEC's normal announcement and distribution procedures may be followed.
- [ ] b. Make available only within AEC and to AEC contractors and other U.S. Government agencies and their contractors.
- [ ] c. Make no announcement or distrubution.

REASON FOR RECOMMENDED RESTRICTIONS:

SUBMITTED BY: NAME AND POSITION (Please print or type)

Lakshmi N. Goyal
Research Assistant

Organization
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Signature	Date
Lakshmi N. Goyal	November 24, 1970

FOR AEC USE ONLY

AEC CONTRACT ADMINISTRATOR'S COMMENTS, IF ANY, ON ABOVE ANNOUNCEMENT AND DISTRIBUTION RECOMMENDATION:

PATENT CLEARANCE:

- [ ] a. AEC patent clearance has been granted by responsible AEC patent group.
- [ ] b. Report has been sent to responsible AEC patent group for clearance.
- [ ] c. Patent clearance not required.