5.5.6.4 DCB Locations CSECT Name: EVQFILES Functions:

1) Called by main CSECT at initialization time,

2) Initializes DCB variables,

3) Stores address of full DCB address list for use by ASSIGN command with DCBLIST DSECT stored in DCBSB in QUESLIST DSECT.

4) Opens system command input/print files and scan file.

Files Allowed

DCB Name	DDNAME	Access Method	DCB Info(RECFM,BLKSIZE,LRECL)	MACRO
SY1	TERMINAL	BSAM	F,8Ø,8Ø, 1 BUFFER, INPUT	NORMAL DCB
SY2	SYSPRINT	QSAM	F,8Ø,8Ø, 1 BUFFER, OUTPUT	QDCB
SY3	MASPRINT	QSAM	FB,32ØØ,8Ø, 1 BUFFER, OUTPUT	QDCB
SY4	SYSUDUMP	BSAM	VBA,882,125, REGULAR DSNAP DCB	DCB
SCY1	EVQSWKØ1	FSAM*	F,8Ø,8Ø, UPDAT	SCANDCB**
SCY2	EVQSWKØ2	FSAM	F,8Ø,8Ø, UPDAT	SCANDCB
W2***	EVQSWKØ2	QSAM	F,8Ø,8Ø	QDCB
W3	EVQSWKØ2	QSAM	---Left to generation time.	QDCB
W4	EVQSWKØ2	QSAM	---Left to generation time.	QDCB
W5	EVQSWKØ2	QSAM	---Left to generation time.	QDCB

*NOTE: The scan file DCB's are between the system output file, which are normally open and the system work files, which are normally closed; this allows comparing the WORKOUT/MASPRINT addresses with the SCANF address. If the former is larger,then do no close them after use; if not, then do so. An open SVC issued on an open file has no effect, thus no checking is done.

**SCANDCB is a special MACRO stored in ECOL.MACLIB.

***Files W2 and below are opened as required and closed after use; all above W2 are opened in EVQFILES CSECT and left open.

Table 5.14 File Definitions

Default Files

SYSIN -- SY1	SCANF -- SCY1
SYSPRINT -- SY2	WORKIN -- W2
MASPRINT -- SY2	WORKOUT -- W2

Table 5.15 Default Files

Full DCB Address List (Used by ASSIGN command)

DCBLIST DSECT	FILE'S ADDRESSES (1 full word each)
PARMA	DUMMY
PRNTS	SY2,SY3,SY2,DUMMY,DUMMY
SCNS	SCY1,SCY2
WKS	W2,W3,W4,W5,DUMMY,DUMMY,DUMMY

Table 5.16 DCBLIST DSECT

5.5.6.5 ASSIGN Command Logic CSECT Name: EVQASIGN

Letter	Return	Offset	Name	Description	DCB Name	USER Name
I	2	0	IZ	Assigns value to	WORKIN	(INPUT)
O	2	4	OZ	Assigns value to	WORKOUT	(OUTPUT)
P	2	8	PZ	Assigns value to	MASPRINT	(PRINT)
S	2	12	SZ		SCANF	(SCAN)
?	1	0	QUES	Dummy to find out assignments		

IA,OZ,PZ,SZ:

LOAD REG2 with A(Appropriate DCB LIST)

Load Reg3 with A(DCB variable to be changed.)

Load Reg5 with A(Position in message line to indicate new assignment.)

Branch to AREST after each.

AREST:

If PARM = C'T'; move A(SY2) to address in REG3; put 'T' in MSG. line.

= C'P'; move A(SY3); put 'P' on MSG. line.

If PARM = a one digit number; convert it to internal form; use the value to index the DCB list pointed to by REG2; move the address found to the address pointed to by REG3, and store the input digit in MSG. line.

AGO: (GO ROUTINE)

Print message line (named FILELN) to the terminal. This CSECT uses DCBLIST DSECT to address the files and initialize DCBASE (REG7) to contain the appropriate base address--it is stored in DCBSB in QUESLIST DSECT by EVQFILES.

5.5.6.6 File Scanning A special access method was used to speed access to the master data file. It uses a hardware feature of the IBM 2311 Disk Packs, in use at Rensselaer Polytechnic Institute, to access an otherwise normal sequential file. FSAM (File Scan Access Method) was developed by John Fisher of the Fresh Water Institute and is only usable on special models of 2311's (which, as mentioned, those at R.P.I. are). FSAM is given a scanning pattern by the managing routine and, starting from the last record matched, finds the next record to match the pattern.

The SCAN command makes use of FSAM to access the master data file which was set-up to allow its use. FSAM requires the following restrictions:*

1) unblocked 80 byte records,
2) one extent,
3) allocation by cylinders, and
4) the first record of the file is blank.

FSAM is not available in source form and so was link-edited into EVQS.LINKLIB from ECOL.LINKLIB. It requires special DCB's which the MACRO 'SCANDCB' will generate. It is in ECOL.MACLIB. FSAM is invoked by linking, i.e., LINK EP=FSAM. A special control block must be passed to it through REG 1.

*A new version without these restrictions was being implemented; it should be complete by now.

```
(1)     FDCB      DS   A(SCAN file SCANDCB)
        PATLEN    DS   H'LENGTH OF PATTERN, starting at byte 1'
        DATLEN    DS   H'LENGTH of data portion'
        PATTERN   DS   A(Pattern)
        DATA      DS   A(Data record)
        MBB       DC   XL8'Ø'
```

Table 5.17 FSAM Control Block

MBB, in the control block, is zeroed initially, and after return from FSAM whenever no records were matched; otherwise it contains the track address of the record found and should not be changed. That action resets the file pointer; zero effectively rewinds the file.

A file may be read both with FSAM and QSAM; providing each QSAM usage is surrounded by opening and closing of the QSAM DCB and the FSAM DCB is initially opened and MBB zeroed before each usage. File 2 is handled in this manner to allow working with a subset of the master data bank.

The pattern is filled either with EBCDIC characters to be matched, or X'FF' in characters where no match is to be attempted. The pattern is eleven characters (bytes) long and the meaning of each is below:

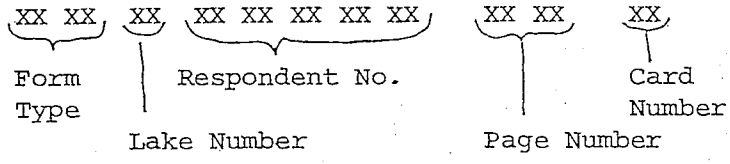


Figure 5.9 SCAN Pattern Format

(X represents a hex digit, XX a character.)
(The pattern is the first eleven bytes of each card.)

5.5.6.6.1. SCAN Command Logic CSECT Name: EVQSCAN

Letter	Return	Offset	Name	Description
F	2	Ø	SF2	Form PARM.
R	2	4	SR2	Respondent Number--only one this run.
	3		SR3	Single value.
	4		SR4	Range.
L	2	16	SL2	Lake PARM.
P	2	20	SP2	Page Number.
C	2	24	SC2	Card Number.

SF2,SL2,SC2,SR2: Load address of where parameter goes in the key (or pattern) in REG2; branch to SREST.

SREST: Use Execute (EX) on a move character (MVC) to move PARM to pattern.

SR3: Move single respondent number to RVALS: fill second number with X'FF'.

SR4: Move both respondent numbers in the range to RVALS.

Variables (in QUESLIST DSECT)

RCNT--F--A(Highest pair of respondent numbers in RVAL)

(=RVAL-1Ø if no resp. number specified.)

RVAL -- 16Z1Ø -- 8 pairs of zoned number five digits each, each pairs in a range of respondent numbers. String (80 chars long) is initialized to 8ØX 'FF' each time SCAN is entered.

MODPAT -- Contains pattern with respondent number filled with first one in range being processed.

SGO: (Go Routine) Read form SCANF using FSAM Link. Write on WORKOUT using DOIO MACRO.

If no respondent number ranges specified:

SGO2: Read ↺ Count
Write
↓ No Match

SGO2NO: Print MSG giving number of records matched.

Exit to SFINE.

Ranges Specified

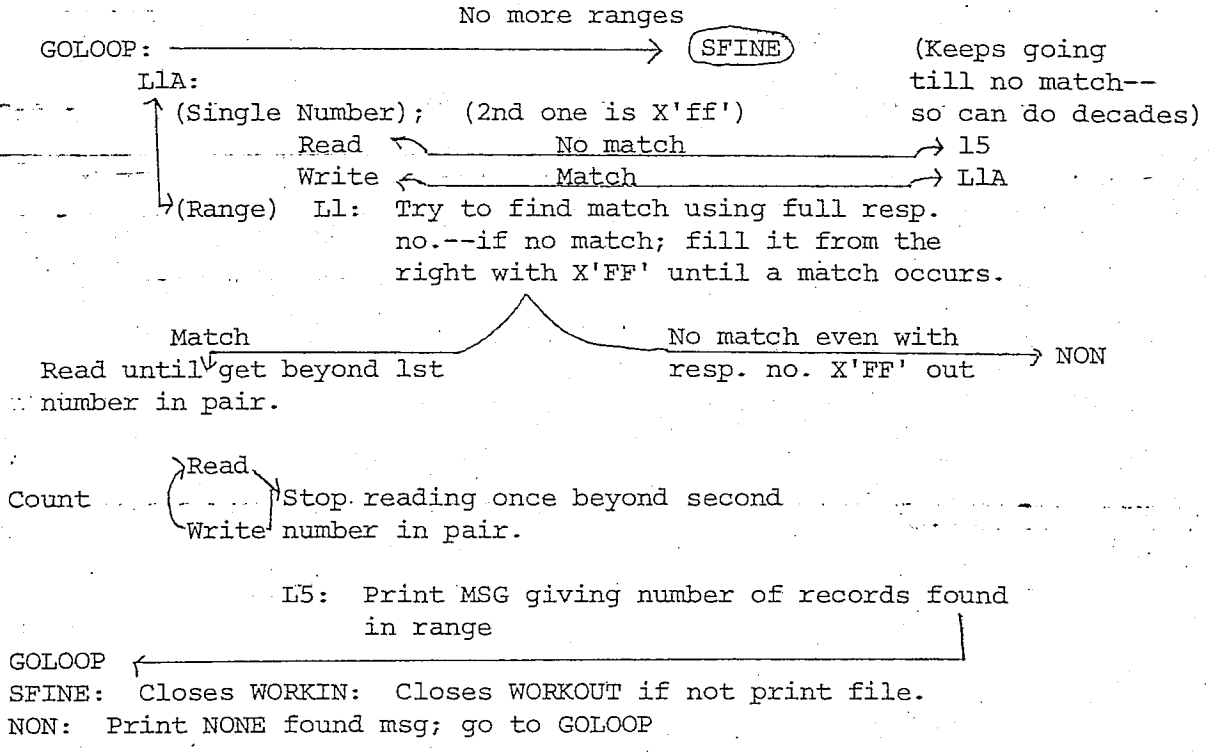


Figure 5.10 SCAN GO Routine Flow Chart

5.5.7 The Analysis Routines These routines consist of 11 sections; show below and in detail in Figure 5.12.

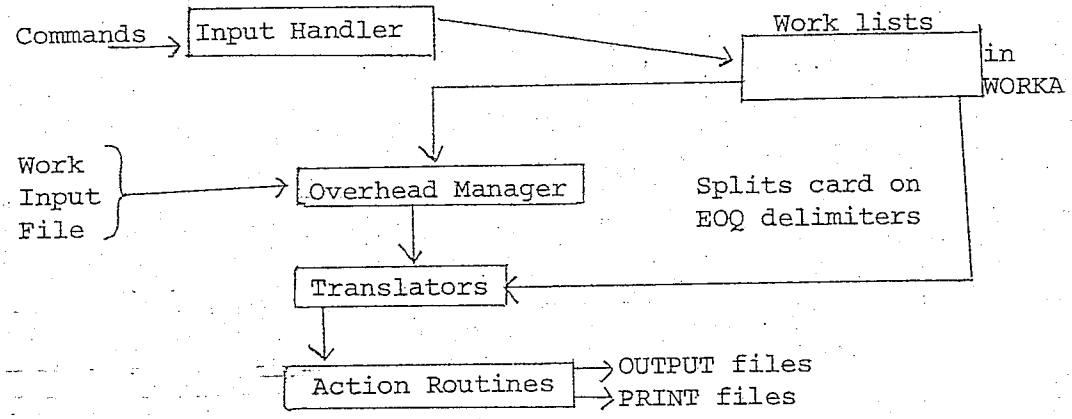


Figure 5.11 Overview of Analysis Routines

The key to the whole system is the use of a work area (named WORKA) to build various lists based on the input commands; which the later routines then use repeatedly to handle each card read from the input file. These work lists, build in WORKA, tell each translator how to handle which question. These lists are linked to QTAB as in Figure 5.13. Since the syntax parser does not indicate the end of a PARM list, the work lists must be build in two steps. 1) build a prototype list as parameters are recognized, 2) then finish the list and fill in pointers to it when a new list is started or processing requested.

In the first step the overhead manager first scans each card according to the information stored in the work lists. The translation routine then converts the input card into bit flags which are stored in the translation work list. Up to eight translation routines (see Table 5.18) allow handling various types of questions. Each translation routine

handles all questions on the card requiring its use. The bit flags also provide a conversion of a complex format into a binary fixed form; and may also be output using the binary list option. Further analysis may be done by other analysis systems.

The conditional routine is the first action routine activated. It checks the translation bit flags of each question requested by the work-lists and combines them according to the logical operators in the work lists. All conditional actions that are active are turned off, if the test fails. All list, tally and binary list requests that are active are then honored by calling the associated action routine.

CSECT Name	Function
EVQCST	Character String translator.
EVQNMC	Numeric multiple choice.
EVQAMC	Alphanumeric multiple choice.
EVQMATCH	Substr match.
EVQRNGE	Range Counter.

Table 5.18 List of Translators

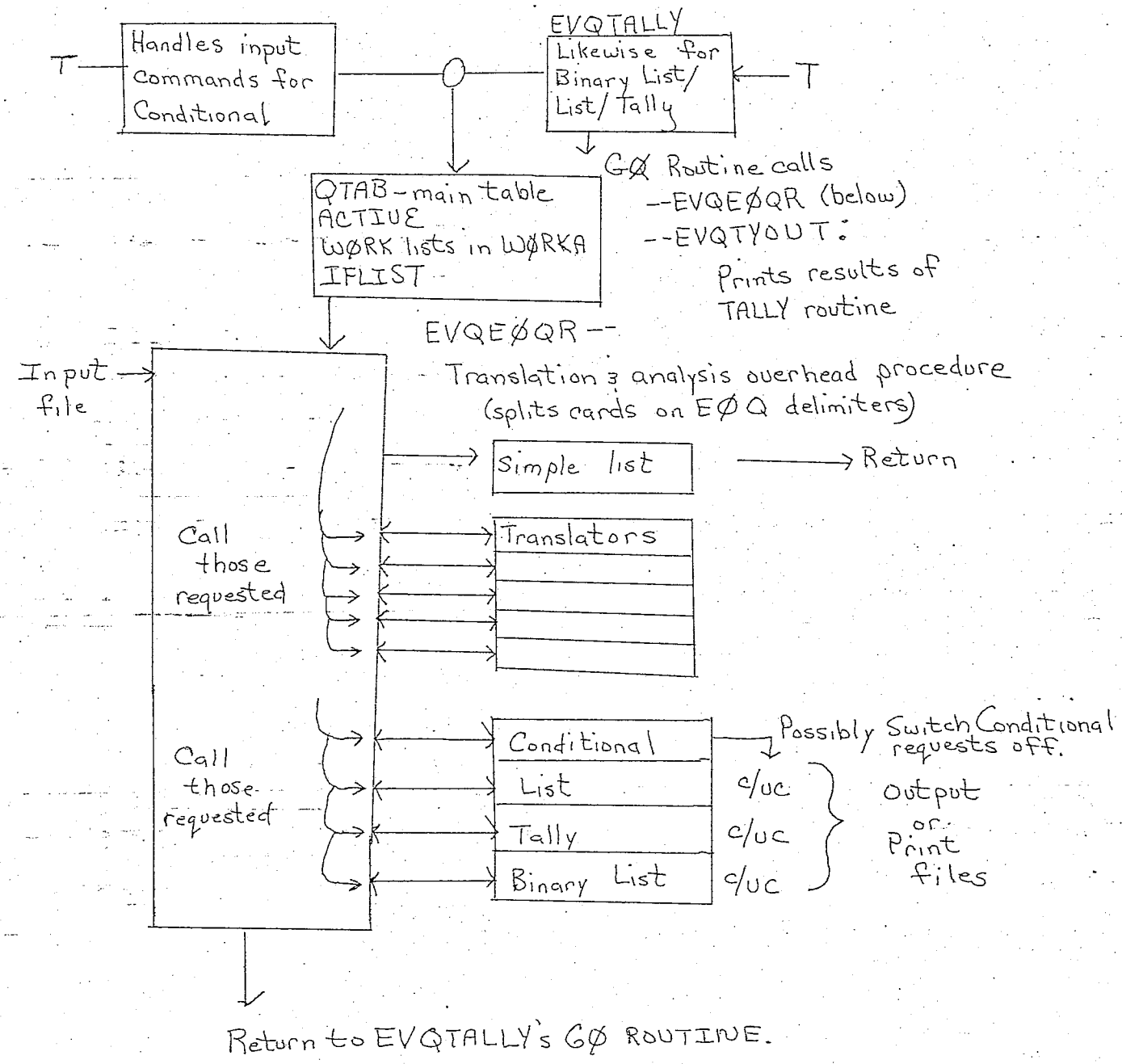
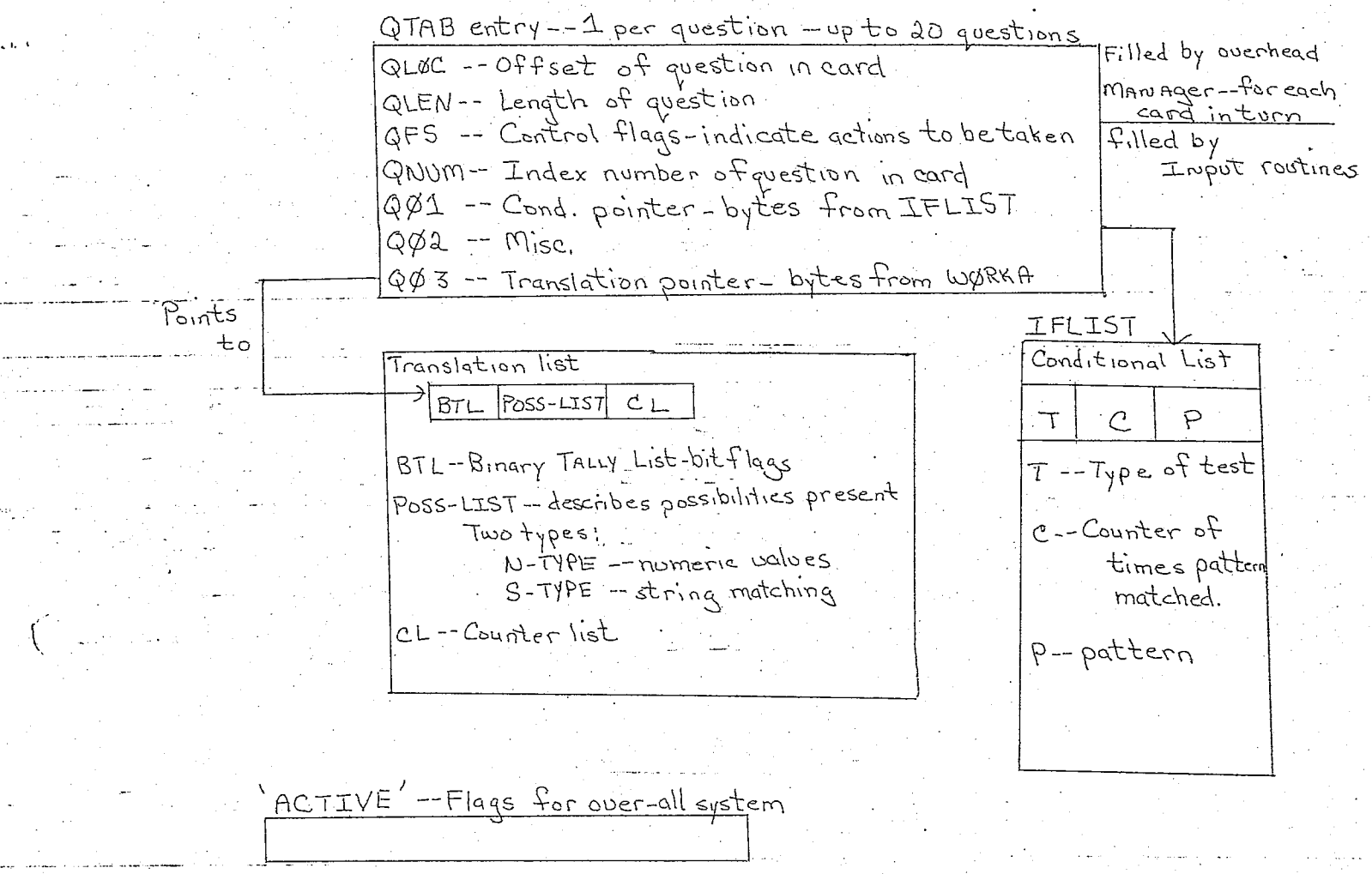


Figure 5.12 Analysis Routines



Each list of flags is fully defined in a subsequent figure.

ACTIVE	-- 4.17	Conditional list	4.20
QTAB	-- 4.18	Basic Translation list	4.21
QFS	-- 4.19	ACTUAL Translation list	4.22

Figure 5.13 Work Lists

5.5.7.1 Translation Work Lists The translation lists are built in three stages:

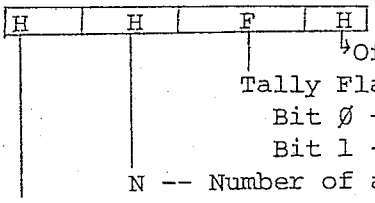
1) Initial stage: This stage is indicated if QO3 of the first question requested, i.e., the one pointed to by Register 2, is zero. The BTL is build using a subroutine--BTLINIT. The possibility list is started; QO3 filled in. Form 2 parameters and the first Form 3 and 4 recognized are all initial operations.

2) Extend stage: This is indicated if QO3 is not zero. BTL is modified as required and the possibility list is extended. All parameters recognized in a Form 3 or 4 PARM list, except the first, are extend operations.

3) Fill Stage: This is indicated when a new work list is started or when a command is completed and the GO Routine is entered. The counter list (CL) is added to the translation list built; copies are made for each question requested and the pointers are filled into the question table, both the translation lists, and the conditional lists.

Storage for the translation lists in WORKA, is managed by a routine, GETSTORE, which also issues warnings and errors as space is used up and exceeded. Aside from BTLINIT and GETSTORE, all initial and extend stage operations are done in-line. Fill stage is done by 'TYFILL' for both conditional lists and translation lists. Lists for user translations, X, Y, Z, are dummied out (TX1,TY1,TZ1). The first two stages, in building the work list, are to build the prototype. This prototype is then used in the fill operation.

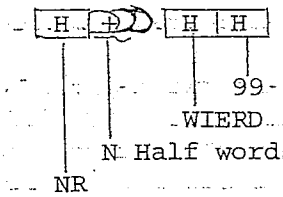
Binary Tally list -- BTL(L,N) Length 1Ø



Initial: GETSTORE space in WORKA -- (Full word boundry),
BTLINIT does this.
Put offset in QO3 for active ques.
Fill in given fields-(set counter address to zero).

Extend: Get location form QO3.
Change fields.

Counter list -- CL(N)** N+3 half words which are counters for corresponding bits.



Counts active flags with left-most bit counter following NR's; and right most bit counter preceding WIERD's.

GETSTORE space in WORKA (half word boundry)
Put offset in BTL.
Zero out N+3 half words.

Possibility Lists These occur in-between BTL and CL. They are placed there because they often store possibility values, which are added as progressive commands are processed. CL, in contrast, is just a zeroed list until actual data is processed.

+Denotes a variable length area.

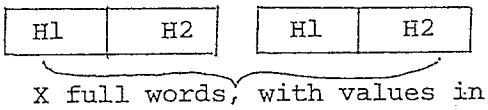
*(The left-most of the N tally bits corresponds to left-most or first counter and first possibility on possibility list.)

**(All translation lists end with CL(N)).

Figure 5.14 Basic Work Lists

Numeric Type

N-TYPE(H1,H2)



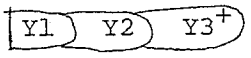
X full words, with values in each half word.

GETSTORE 2 half words

Fill-in fields.

String Type

S-TYPE(Y_N)



N strings of varying length

GETSTORE Y(i) bytes

Store strings

+Denotes a variable length area.

Figure 5.14 Basic Work Lists (cont'd)

5.5.7.1.1 Building Prototypes An INDEX (originally question, hence TQ2,TQ3,TQ4) parameter starts a new question prototype in ACTIVE. Once the prototype is started, its address is stored in REG2STRE. REG2STRE is initialized to the beginning of QTAB and each fill sets it to the next open entry.

As options and translations are specified, bits are turned on in ACTIVE. The offset of the translation worklist are likewise filled in. The prototype is filled for the rest of the indicies when TYFILL is called. This may occur in two places:

1) If not previously done, do so in Go routine, TYGO, but only if hold flag is off; or

2) When a new index parameter is encountered.

The need for the fill operations is signified by the third bit on in TINTFG. It is turned on by the index and off by the fill. The fill also turns the last quesiton bit on. The GO routine turns the last question of last card bit on.

5.5.7.1.2 Building Sequence

1) The first tally preface, without the hold flag set, begins the sequence:

--Zero's ACTIVE, TINTFG, TQOFS, TQNUM, PTYPE, QO3.

--Set QSTORE to point to beginning of WORKA.

--Sets on first question bit in first question's QFS.

--Set REG2STRE to point to beginning of QTAB.

1a) Any tally, with hold on previously, will load REG2 from REG2STRE; each parameter routine using QTAB must also

do so, as a base register for QTLIST DSECT.

2) The FORM parameter:

--Do necessary fill operations--(check flag in TINTFG to see if a true FORM parameter or a INDEX parameter usage, see 3 below.)

--Increment PTYPE and place new form in TFCHK.

--If not first question, then turn previous 'last question of card' bit on.

3) The INDEX parameter:

--Do fill operations, by using a section of FORM parameter code. Set bit in TINTFG to indicate usage by INDEX.

--Place entered values in TQUES.

--Increment TQNUM.

4) OPTIONS, HOLD, FINE, GO:

--Set bits in ACTIVE and TINTFG.

5) Translations:

--Turn bit on in ACTIVE+3.

--Build prototype work list, and store its offset in Q03 referenced by value in REG2STRE as base register.

6) Fill Operation:

--TYFILL-(TALLY)--Use 'GETSTORE' to get space in WORKA.

--Fill out counter list of translation--bomb if no room.

--Decrement TQNUM and expend QTAB. Copy work lists for each question.

--Copy QFS from ACTIVE+2(+3) and copy Q01 from ACTIVE+4 (+5).

--Store next open question entry in REG2STRE and REG2.

--Initialize for next INDEX build sequence.

5.5.7.2 Flag Description Tables 'ACTIVE' is a double word that contains various flags used during each run. It is so arranged that its third and fourth bytes are identical to those of QFS (each question's flags). This is such that all the QFs's may be logically OR'ed to determine what translations and action routines must be called. ACTIVE +2 and 3 are also used as a prototype while QFS is being built. ACTIVE is zero'd during the preface of the TALLY command. Numbers start with the left-most bit as bit Ø.

(Bit)	ACTIVE	Ø - No/1 - Yes	ACTIVE+2	Options
Ø	WIERD	Ignore/SAVE	List--Unconditional (UC)	
1	99	(Same)	--Conditional (C)	
2	ALMOST	(Same) Not implemented.	TALLY--UC	
3	NR	(Same)	--C	
4	Hold Flag		Binary List--UC	
5	Fill Flag		--C	
6	Overall Conditional Test			
7	(See Conditional list description)			

	ACTIVE +1	ACTIVE+3 Translations	
Ø	Conditional	1	CST
1		2	NMC
2	Formating	3	AMC
3		4	MATCH
4		5	RANGE
5		6	
6	Multipart--Any ques.	7	
7	Store len of active ques. flag	8	

ACTIVE +4(+5) is used to store prototype of QOl--half word.

Table 5.19 'ACTIVE' Flags

DSECT for use with QTAB in ADRLIST

*	2D	--L16--Base Register 2
QLOC	H	Location in CARDIN (of previous EOQ)
QLEN	H	Length of actual question (EOQ no counted.)
QFS	F	Flags--See Table 5.22
QNUM	H	Index number of question.
QO1	H	Offset of conditional test in IFLIST
QO2	H	
QO3	H	Offset of translation work list in WORKA.

QTAB is twenty questions long, with each question entry being sixteen bytes. By initializing Register 2 with A(QTAB) and augmenting it by 16, the symbolic names in QTLIST may be used to reference each question entry.

Table 5.20 QTAB Table & QTLIST DSECT

Name	Source
QLOC	EVQEOQR--On processing
QLEN	EVQEOQR
QFS	Most from commands/default
QO1	On fill of IFLIST
QO2	Not used
QO3	On fill of WORKA

Table 5.21 Source of Entries in QTAB

QFS is one full word long.

BIT	QFS	QFS+2
Ø		List --UC
1		C
2		Tally --UC
3		C
4		Binary List--UC
5		C
6		
7		

	QFS+1	QFS+3	TRANSLATIONS
Ø	If 1st question in QTAB	1	CST
1	Multipart ques. (EOP found)	2	NMC
2		3	AMC
3		4	MATCH
4		5	RANGE
5		6	
6	Last ques of a card	7	
7	Last ques of all cards	8	

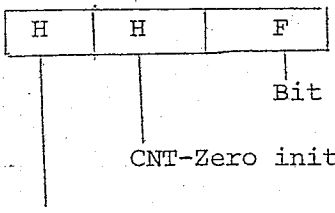
If bit 7 is on, 6 always is also.

Table 5.22 Individual Question Flags, QFS in QTLIST

Both IFNUM and IFLIST are in QUESLIST DSECT.

IFNUM--1 full word has the offset of the highest-used double word in IFLIST.

IFLIST--15 double words--each entry is a double word.



Bits Ø - 4---Always zero

5---Action to take on match: 1-STOP,Ø-GO(default-Ø)

6, 7 } Test to take -- AND ØØ
OR 1Ø
(Exclusive) XOR 11

Flags are such that X'54' OR'ed with them give proper OP code for comparison.

X'54' OR'ed X'ØØ' = X'54' = N (AND)

X'Ø2' = X'56' = O (OR)

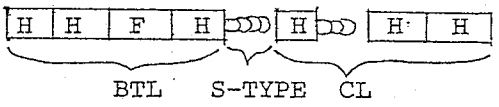
X'Ø3' = X'57' = X (XOR)

Default on individual test is OR; on summing of tests is AND.

Figure 5.15 Conditional List Variable & List

5.5.7.3 Specific Work Lists

CHAR = XX,(XX,XX,XX) Forms



2,3

(TC2)* Initial: BTL(L ← PARML-1,N ← 1) Turn on fill flag.

(TC3A) S-TYPE (Y_1 ← PARM)

(TC3) Extend: BTL (N ← N+1)

(TC3A) S-TYPE(Y_N ← PARM)

Fill: CL(N)

Definitions

BTL L--Length of each possibility minus 1 (All the same length)

N--Number of possibilities (Incremented as each is processed)

S-TYPE (N possibilities of length L+1 each)

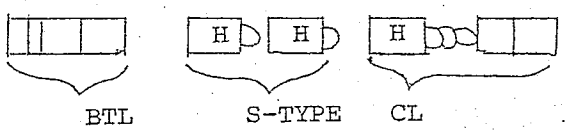
Y_N--Current possibility being processed and stored.

*Names in parenthesis are the labels of these steps.

Figure 5.16a Specific Work lists--CHAR

MATCH = XX,(XX,XX,XX)

Form 2,3



(TM2) Initial: BTL(L ← PARML-1,N ← 1) Turn on fill flag.

(TM3A) S-TYPE (Y_N ← (PARML-1,PARM))

(TM3) Extend: BTL(L ← MAX(PARML,L),N ← N+1)

(TM3A) S-TYPE(Y_N ← (PARML-1,PARM))

Fill: CL(N)

Definition

BTL L---Maximum Length -minus 1 (So must determine largest so far)

N--Number of possibilities

S-TYPE (N possibilities of length ≤ M+2)

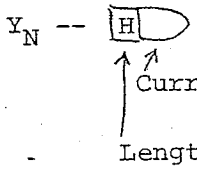


Figure 5.16b Specific Work lists--MATCH

NMC = XX

Form

2

TN2

(TN2) Initial: BTL(L← 0,N ← v(PARM))

Fill: CL(N)

Definition

BTL L---Not used

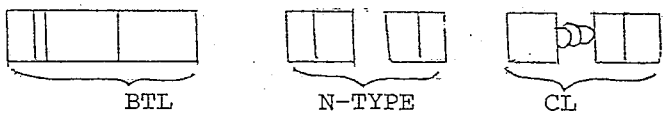
N---Number of possibilities

v(PARM)---numeric value of PARM character string

Figure 5.16c Specific Work lists--NMC

AMC = (XX,XX-X,XX-X) Turn on fill flag

3,4



(TA2/TN2) Initial: BTL(L← 0,N ← v(PARM))--Save as NMC

(TA4) Extend: BTL(L ← L+1,N ← N+H2)

N-TYPE(H1 ← PARM+4,H2 ← PARM)

Fill: CL (N)

Definition

BTL L---Number of choices with letter suffixes

N---Total number of choices

N-TYPE H1--- -Number with letter suffixes-2 digits

H2--- -Number of suffixes -1 digit

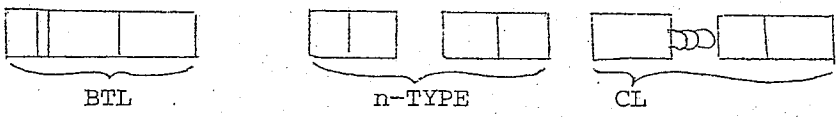
X---L choices, therfore L full words

Figure 5.16d Specific Work lists--AMC

RANGE = (XXXXX-XXXXX,XXXXX-XXXXX,XXXXX)

Form

3,4



(TR4) Initial: BTL(L ← 0,N ← 1)

(TR4A) N-TYPE(H1 ← PARM+4,H2 ← PARM)

Extend: BTL(N ← N+1)

(TR4X7ND) N-TYPE(H1 ← PARM+4,H2 ← PARM) Same as initial.

Fill: CL (N)

(TR3) Form 3: PARML ← 2,PARM ← A(H'00') Continue as if Form 4.

Definition

BTL L---Not used.

N---Number of ranges

N-TYPE (N possibilities of 2 half words each)

H1---First number in range

H2---Second number in range

(H2 ← H'00' for Form 3(single number))

X---N ranges

Figure 5.16e Specific Work lists---RANGE

5.5.7.4 IF & TALLY Input Handlers

5.5.7.4.1 Logic Tables

IF : CONDITIONAL INPUT HANDLER
CSECT NAME : EVQIF BASE = BR1 LETTERS = 1Ø

Order	Letter	Returns	Offset	Name	Description
3	T	3	(Ø) fudged 52	IFT3	Build test image, or set flags
		4		IFT4	Build test image
		2		IFT3	As above
7	A	1	4Ø	IFA1	Set flags in active
8	O	1	44	IFO1	Likewise
9	X	1	48	IFX1	Likewise
1Ø	H	1	52	TH1	See tally
1	G	1	Ø	TG1	Command logic
6	F	1	32	TF1	
		2		TF2	Operations used
4	P	2	12	TP2	by both to
5	I	2	16	TQ2	specify
		3		TQ3	questions
		4		TQ4	requested.
2	N	2	Ø	IFN2	Use previously defined image.

Table 5.23 IF Input Handler

TALLY : INPUT HANDLER
CSECT NAME : EVQIF LETTERS = 17
ENTRY NAME : EVQTALLY, (EVQTT1=USE TO INITIALIZE TABS)
BASE = BR1

	Returns Letter	Active	Offset	Name	Description
3	P	2	12	TP2	Set-up page check list.
2	I	2	Ø	TQ2	Set vector, TQNUM=1
		3	Ø	TQ3	Extend vector, add 1 to TQNUM.
		4	Ø	TQ4	Expand range, add number to TQNUM.
5	O	2	2Ø	TO2	Set appropriate flag.
		3		TO2	Based on number or letter.
12	T	1	76	TT1	Set tabs to default (EVQTT1).
		2		TT2	Set tabs all to equal value.
		3		TT3	Set tabs in order.
		4		TT4	Set requested tabs now.
1	H	1	Ø	TH1	Turn hold flag on.
13	G	1	92	TG1	Turn hold flag off.
4	F	1	112	TF1	Turn hold flag off and stop processing.
		2		TF2	Set-up page checklist--same as P2.
11	M	2	68	TM1	
		3		TM2	
9	R	3	44	TR3	
		4		TR4	Build respective
7	C	2	36	TC2	Work lists
		3		TC3	
8	N	2	44	TN2	
10	A	3	52	TA3	
		4		TA4	
6	S	3	24	TS3	Set bit for NE.
		4		TS4	Set bit for 99 & WIERD.
14	D	1	96	TD1	Display all requests. If not active, use PDUMP.
15	X	0	1ØØ	TX1	
16	Y	0	1Ø4	TY1	Excess for last 3
17	Z	0	1Ø8	TZ1	translators (dummied out).

Table 5.24 TALLY Input Handler

5.5.7.4.2 Variables

Tally Initial Flags TINTFG

on

*IF flags Ø - Test later bits

1 - Form 4 = 1 --Used by test to use Form 3 code

2 - Fine' --Terminate

Tally flags 3 - Needs filling

4 - Use by form filling section for index command.

PTYPE -- Number of pages to check.

TFCHK -- Page check list (in QUESLIST) -- entry.

TABGUARD -- Column guard band.

TQNUM -- Number of questions to be filled.

TQUES -- List of questions to be filled.

REG2STRE* -- Contains A(prototype) once started--filled in during command preface.

5.5.7.4.3 IF Input Logic

EVQIF: Begin.

Declare tally external, and set REG2 to QTAB.

Load COMS address.

Reset base address to start of tally section.

Zero IFNUM.

IFT3: First entry; take offset in IFNUM and store in ACTIVE+4.

*For parameter handling entires requiring QTAB table and QTLIST DSECT, one must load REG2 from REG2STRE at beginning on entry. This is because REG2 is used for work register by syntax parser and so must be reset.

Issue name message.

Set requested bit.

Turn on IFFILL flag in TINTFG.

Later entries: (until filled-offset is in IFNUM)

IF S -- Set flag in IFLIST	1	
G --	Ø	Default
IF A, O, X -- Set flag in IFLIST	1Ø	
	Ø1	Default

IF number, --- Turn on that bit.

IFT4: Turn on range; Use IFT3 and bit 2 of TINTFG.

IFA1 ---, IFO1 ---, IFX1 --- Set flags in active	1Ø	
	Ø1	
	11	

IFGO: Print test number.

Go to TYGO.

IFN2: Store input number in ACTIVE+4.

5.5.7.4.4 TALLY Input Logic

START: Load REG2 with REG2STRE --A(PROTOTYPE)

Load command tables.

Return IF hold flag set.

Zero H ACTIVE+4 number of pages.

Zero TQNUM.

Zero fill-flag, PTYPE.

Zero ACTIVE+1 End.

TF2: Set-up page check list (TFCHK).

TP2: Check fill-flag if on, then fill both TYFILL and IFFILL.

Check PTYPE 4 --if so, too many pages.

(PTYPE) 0 --- Set last question of page flag.

Add 4 to PTYPE.

Also set REG2 at beginning of first question of page.

TQ2: TQNUM ← 1 Check fill flag.

TQUES(1) ← PARM if on-- B TYFILL

TQ3: TQNUM ← 1+TQNUM

TQUES(TQNUM) ← v(PARM) (v(x) is numeric value of x)

TQ4: CNT ← v(PARM)-v(PARM+4)+1

BCT CNT

TQNUM ← 1+TQNUM

v(PARM+4) ← v(PARM+4)+1

TQUES(TQNUM) ← v(PARM+4)

TO2: Set bits by number ACTIVE+2

Ø LIST -- UC

1 C

2 TALLY --UC

3 C

4 BINLIST--UC

5 C

TT1: Calculate default tabs; use TT2A with REG3=8.

TT4: Use v(PARM+4) as offset in TABSS, to store half v(PARM) as width

Recalculate TABSS totals from there on.

TH1: Turn hold flag on.

TG1: Turn hold flag off.

TT2: (TT3 does same). Convert PARM to internal value REG3.

TT2A: Calculate value of tabs given tab width in REG3.

TS3: Set NR flag in ACTIVE.

TS4: Set WIERD and 99 flags in ACTIVE.

} =Ø if ignore
1 if save

TD1: Interpret all falgs and such, use PDUMP.
Was to be 'DISPLAY' option, not implemented.

TF1: Turn hold flag off and fine' flag on.

3.4.7.4.5 Storage Mananger

Storage Routine

GETSTORE -- passed length request in REG1.

Start at location QSTORE.

Test if QSTORE+VALUE requested by test + 1 SYSIN.

IF = issue warning - 4 (in REG 15)

IF issue termination message +8 (in REG 15)

IF ;request if O.K., return with new ½ word aligned address

in QSTORE: Ø in REG15

& startingaddress of space allocated in REG1.

IF 'F' request--test QSTORE for Fboundry extract

last CHAR--see IF Ø,4,8,C;

Negative number if F request, so test for zero

if so add ½ word to QSTORE to align to full word boundry.

TYFILL FILL-IN ROUTINE

CL(N).

L(IMAGE)=QO3+A(WORDA)-QSTORE.

GETSTOREL(K+IMAGE).

} (Covers only one card.)

PUT offset in QO3.

MVC image in.
Fill QFS from ACTIVE. } (Covers only one card.)

Put in QUES number in QNUM; for each ques in TQUES vector.

Finished by setting last ques QFS flags.

Start with REG2 at first of req. sequence.

TYGO: Check fine' flag.

Yes--Hold flag off, return.

If hold flag set return.

Call fill in routine.

Back up one.

Set last question of all flag.

Call EVQEOQR.

Call EVQTYOUT to do output.

Percentage -- { In item <- total tally
Absolute -- { of respondent <-

Sample size.

5.5.7.5 EVALUATIVE ROUTINES — EVQEOQR

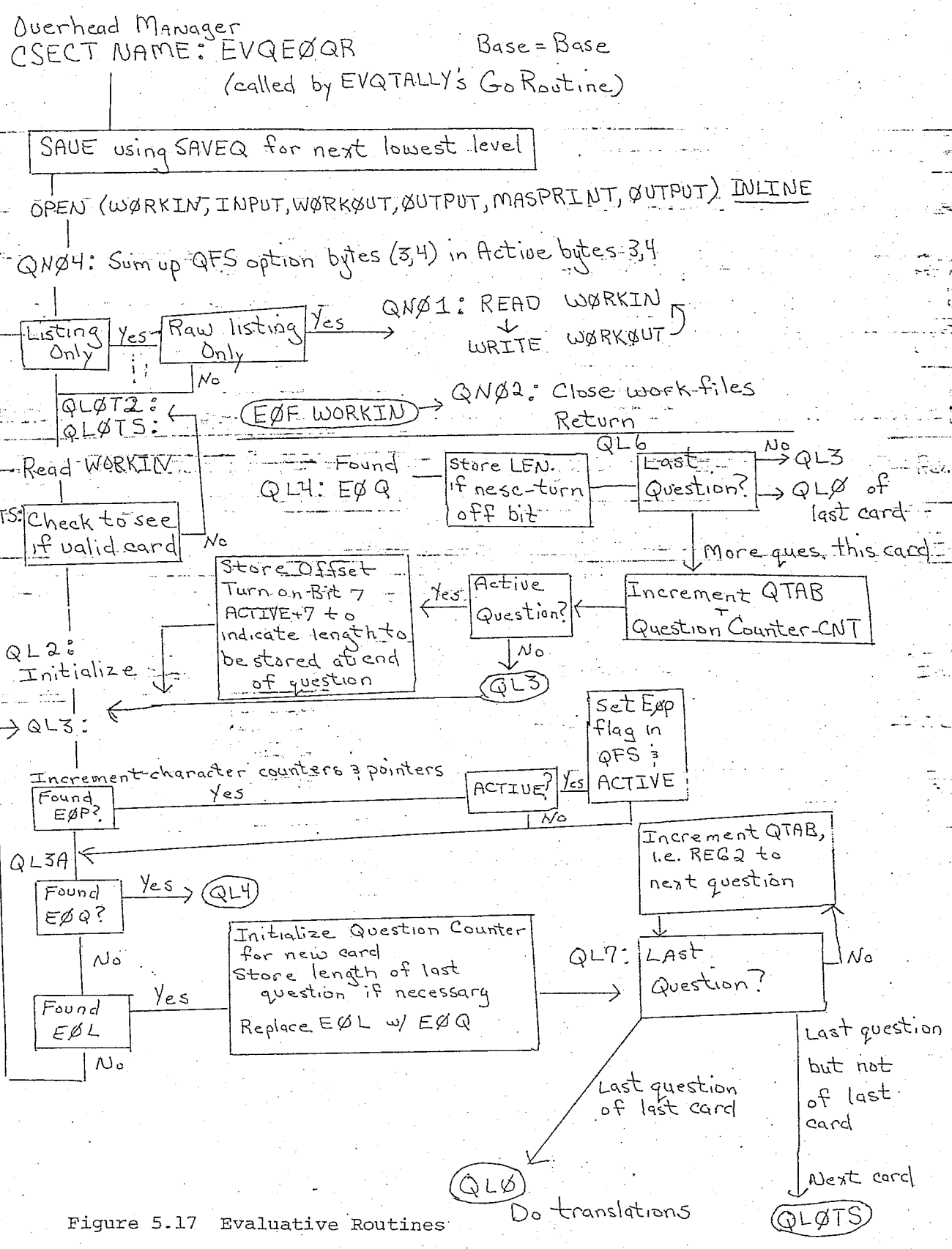


Figure 5.17 Evaluative Routines

5.5.7.5.1 Action Routines

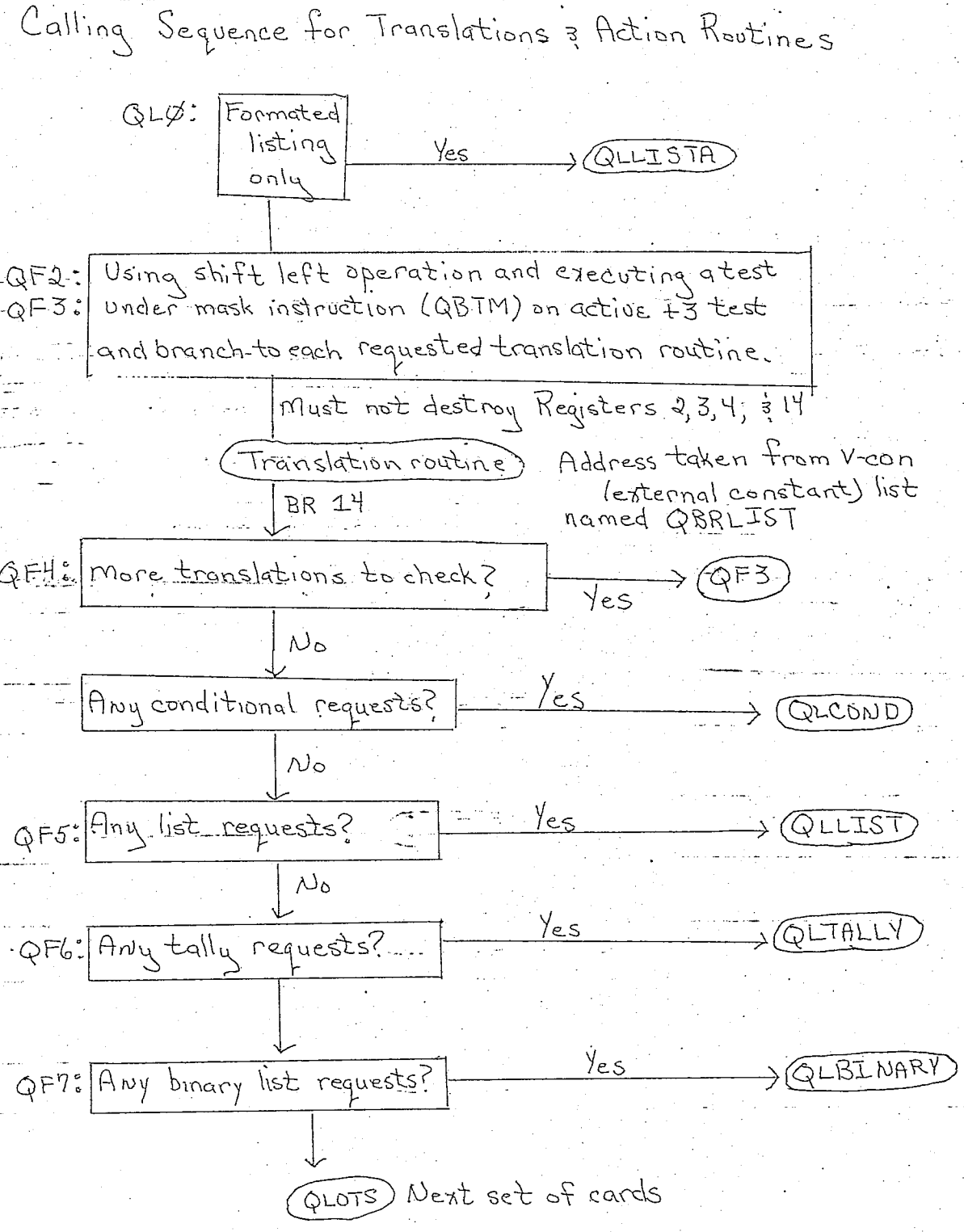


Figure 5.17 Evaluative Routines (cont'd)

5.5.7.5.1.1 QLLIST -- List

LIST ACTION ROUTINE

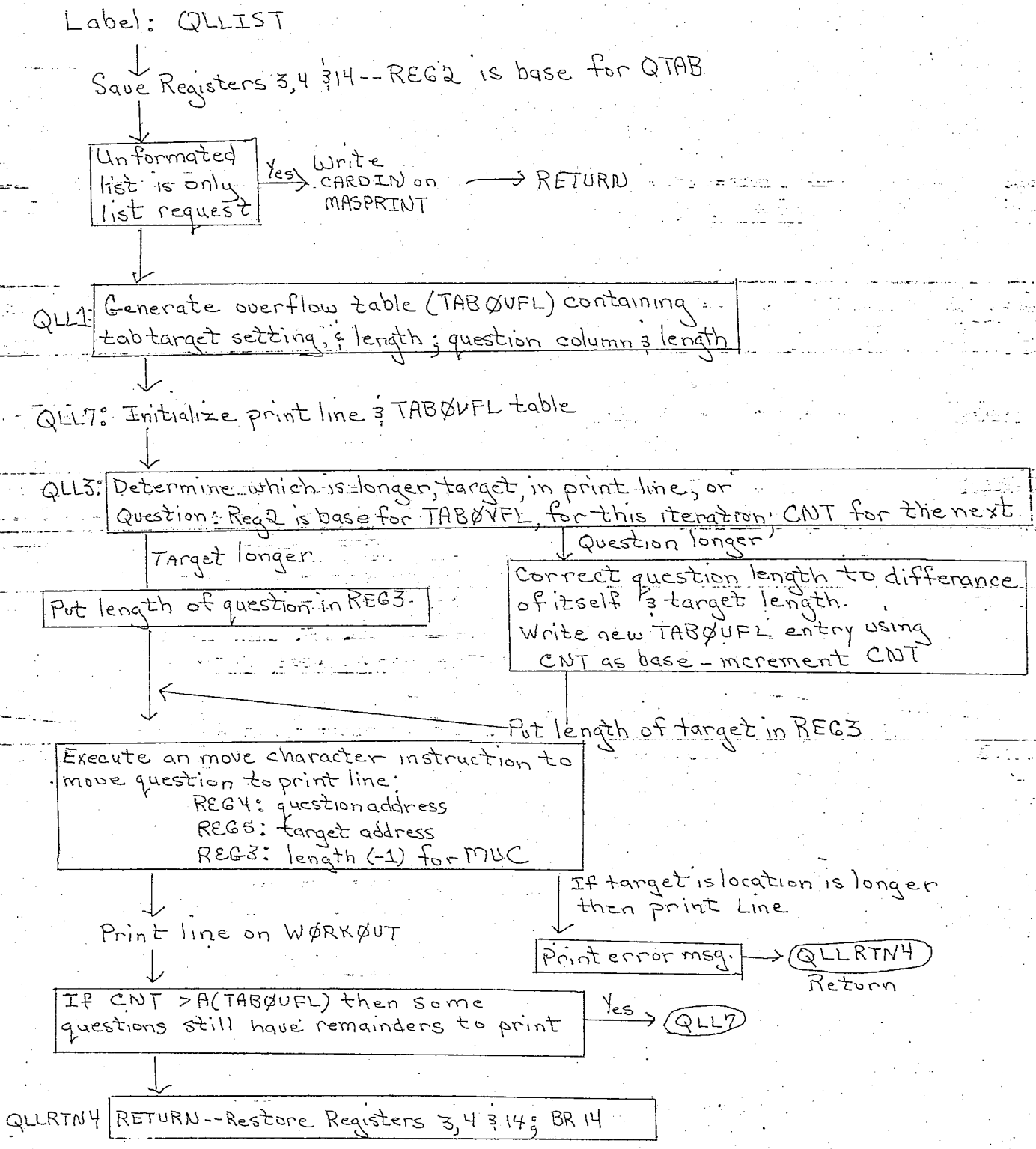


Figure 5.18 List Action Routine

5.5.7.5.1.2 QLTALLY-Tally

QLTALLY: (entry) BINARY TALLY ACTION ROUTINE

Uses BTL at offset QO3 from WORKA.

And C1 whose offset is stored in BTL.

Save

QLTT1: Initialize counters and base registers so

CNT--Number of counters/bits to be checked

REG5--Bit flags-always do logical test

so REG5 is not changed.

REG4--A(1st counter).

Test NR (no response) bit if on, tally TNOØØ return.

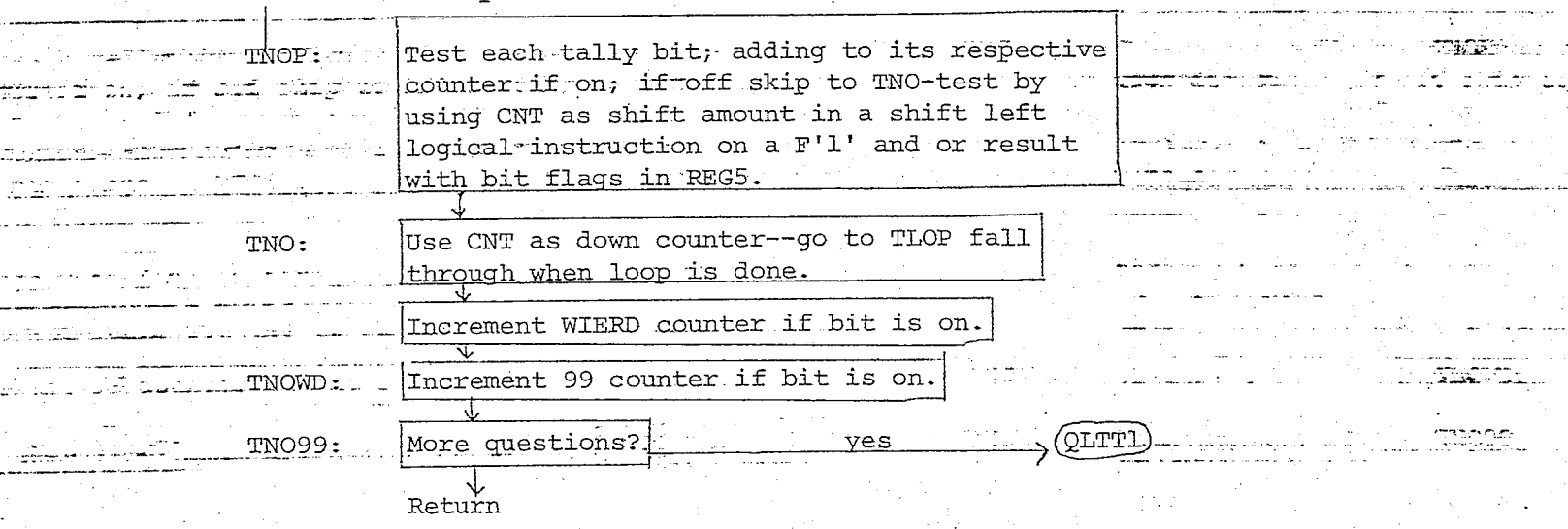


Figure 5.19 Tally Action Routine--QLTALLY

5.5.7.5.1.3 QLBINARY--Binary List

QLBINARY: (entry) BINARY LIST ACTION ROUTINE

Uses BTL offset QO3 in WORKA.

And outputs to MASPRINT (PRINT)

↓

SAVE

↓

TBIN2: Initialize counters & base registers

BR--tally flags

CNT--31-down counter & shift argument

↓

QBIN1: Loop to test bits Ø - 3. & for all that are on put C'1' in appropriate place in QLBLIST (PRINT LINE)--left most bit becomes right most one on print line.

QBIN3:

↓

Test & fill for Bit 31

↓

QBIN4: Move question number and number of poss. to print line in printable form;

Print line on WORKOUT

↓

More question? —yes→ QBIN2

↓ No

↓

RETURN

Figure 5.20 Binary List Action Routine--QLBINARY

5.5.7.5.1.4 QLCOND

COND EVALUATION ROUTINE -LOGIC

(requires work, esp. QLCD1 section)

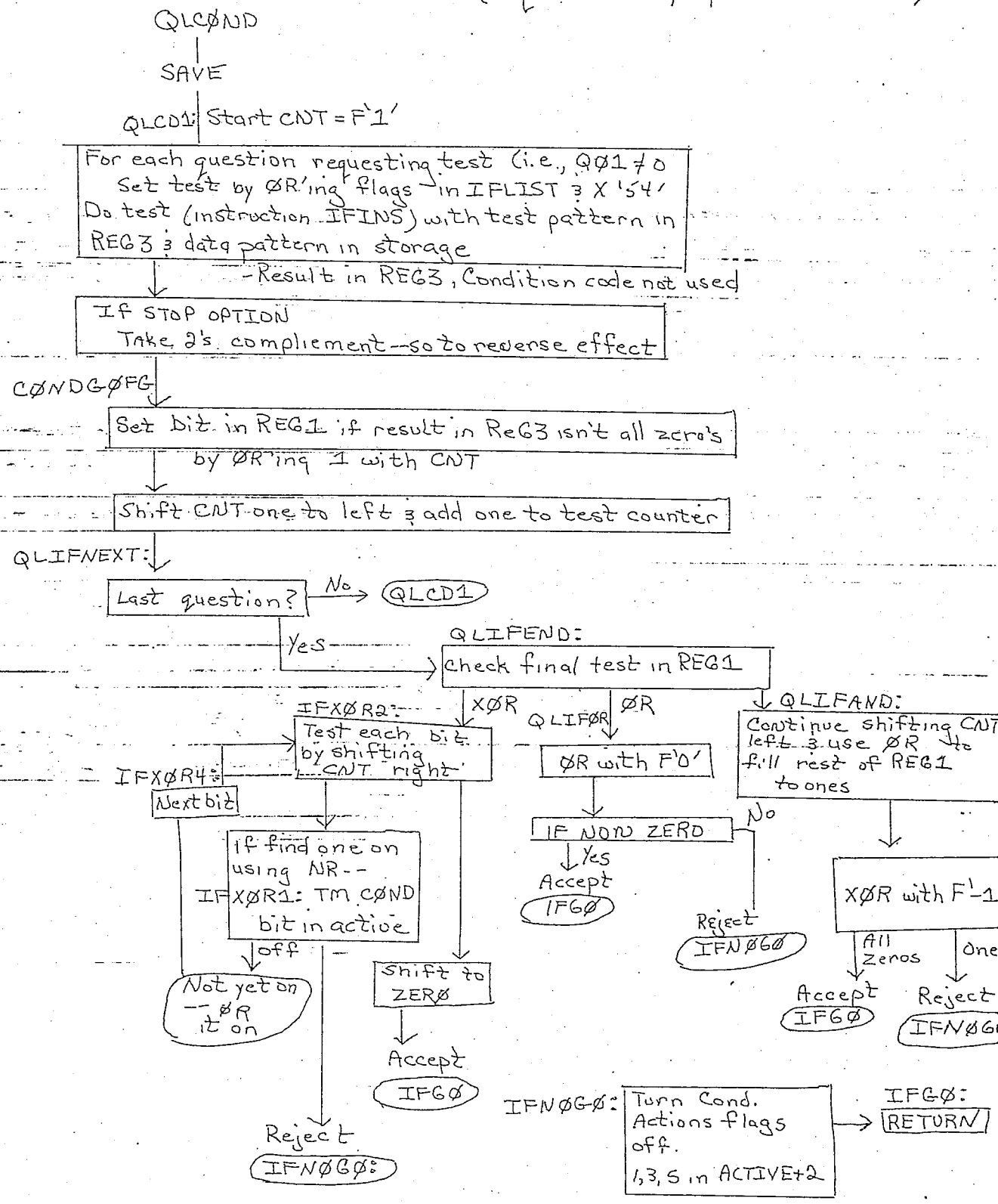


Figure 5.21 COND Evaluation Routine--Logic QLCOND

5.5.7.5.2 Evaluation Routine

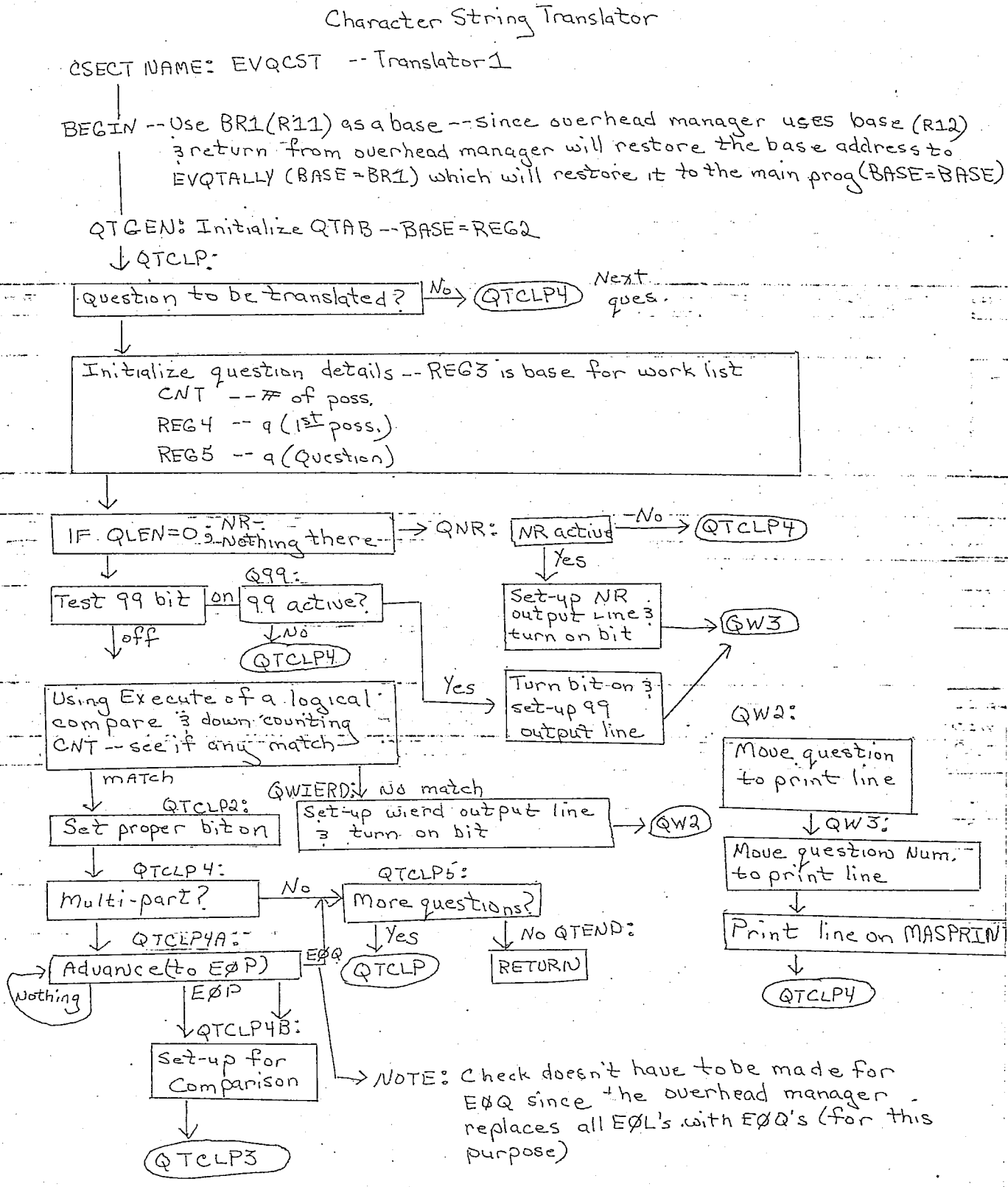


Figure 5.22. CHAR Translator--EVQCST

5.5.7.6 Analysis Outputs The analysis routines have a number of results, both normally because of a request, and when a special case is found. To standardize the handling of these, a number of conventions were developed:

1) Normal outputs go either to the terminal, if small enough; or to the file assigned to WORKOUT (OUTPUT).

2) Special cases go to MASPRINT if active and are ignored otherwise. If a tally is requested they are tallied regardless if they are active or not. The standard format is shown in the Terminal User's Manual. (Section 3.3.7.1)

Function Outputs

Request	File	ASSIGN UNIT	Done by
TALLY	SYSPRINT	(SYSPRINT is output on the terminal.)	EVQTYOUT
LIST	WORKOUT	OUTPUT	QLCOND
ERRORS	SYSPRINT	---	Where Occurs
BINARY LIST	MASPRINT	PRINT	QLBINARY
TRANSLATION: WIERD,99,WIERD	MASPRINT	PRINT	Translator

Table 5.25 Function Output Units

5.5.7.6.1 TALLY OUTPUT ROUTINE--EVQTYOUT

CSECT : EVQTYOUT BASE = BASE

OUTPUTS:

1) Tally counters for each question; and any test used; and what was requested.

2) Counters and patterns for each test.

PRINT FORMAT: (with column numbers in parenthesis')

FORM AND PAGE: XXXX
(16)

QUES	Requests	Trans.	Test	N	Counters			
XX	XXXXXXXX	X	XX	XX	XXX	XXX	XXX	...
(1)	(6)	(15)	(21)	(26)	(28)	(32)	(+4)...	
						(32 counters)		

OVERALL TEST: XXX
(17)

Test	Logic	Action	Counter	Pattern
XX	XXX	XXXX	XXX	XXX
(1)	(6)	(12)	(21)	(28)...

Possible logic--uses three functions:

TYOA--Converts bit pattern to printable form,

TYOF--COnverts logic flags to words,

(MACRO) EPCVAL--Converts number to printable form.

PRINT Logic

Convert	Label	Function Used	Notes
Each form	TYOB	Simple MOve	From TFNUM
Each question	TYOC		
'QUES NUM'		EPCVAL	
'REQUESTS'		TYOA	
'TRANS'	TYOD	EPCVAL	Use shift/loop to convert trans. flag to number
'TEST'		EPCVAL	
'N'		EPCVAL	
'COUNTERS'	TYOG	EPCVAL	Loop for all counters
'OVERALL TEST'		TYOF	From active
TESTS	TYOH		From IFLIST
'TEST'		EPCVAL	
'LOGIC'		TYOF	
'PATTERN'		TYOA	
'COUNTER'		EPCVAL	
'ACTION'		TYOHA	Test under mast of IFLIST flags.
NEXT TEST		TYOHB	

5.5.7.6.2 FORTRAN Print Routine--EVQFORPT

Fortran approach: EVQFORPT

A Fortran program is passed the address of the three sections:

QTAB, WORKA, IFLIST. Each is dimenstioned as a Fortran integer*2 array and outputting is done using Fortran's formating ability. The calling sequence is standard Fortran. The only special routine required is one-called from Fortran--which returns a integer array when passed a bit string. This is need to convert the bit flags to printable form--EVQFBITS.

5.5.8 Documentation Function--QUERY This was supplied to allow the terminal user to have access to various useful listings. It is not possible or feasible to access all of the user's manuals from the terminal hence it is advisable to always have them convenient when using the system--but it was deemed necessary to be able to request the terminal for some information. The information was stored in a partitioned data set; each 'query' request builds a member name and searches for it:

Keyword	Member Name Format
FORM/PAGE (both have same effect)	PAGEXXXX
ITEM	ITEMXXXX
HELP	SPXXXXXX

The file associated with the DD card named 'DOCUMN' is the documentation file:

RECFM=FB

LRECL=8Ø

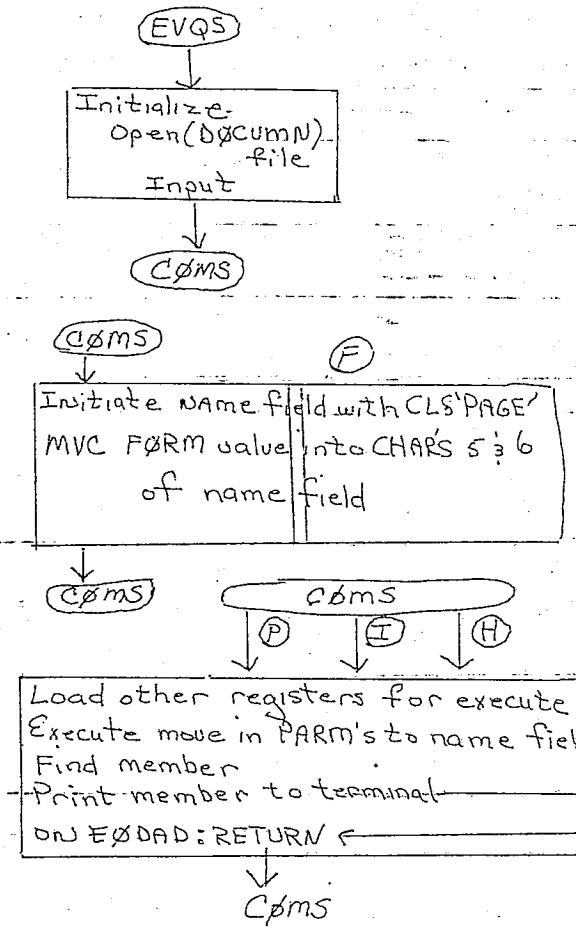
DSORG=PO

BLKSIZE=36ØØ

DCB information for 'DOCUMN' file.

QUERY FUNCTION LOGIC

CSECT: EVQUERY



Number of letters: 4

Letter	Returns Active	Offset (of Return1)	NAME	Desc
F*	2/3	0	QYP2	Form
P*	2	4	QYP2	Page
	3	4	QYP2	
I	2	12	QYI2	Item
	3	12	QYI2	
H	2	20	QYH2	Help
	3	20	QYH2	

* It was decided form & page should have identical effects

DCB for Queryes is DØCUMN & DDNAME is likewise

Member names 'PAGEXXXX'
Form Page
- both are supplied in the same parameter

'ITEMXXXX'
Item

'SPXXXXXX'
Help

Figure 5.23 QUERY Flow Chart

5.5.9 System Programmer's Functions Debugging a system such as this off-line requires an excessive amount of time; on the R.P.I. computer installation, on-line debugging has no provision for easily reading abend dumps. Consequentally a special set of functions were written to allow on-line dumps; both on request and on program interrupt. The functions to do this are described hence:

Command	CSECT	Keywords	Meaning
BEGIN	EVQBGN	ERROR	-Initializes error interrupt handler
		WAIT	-Puts EVQS system in wait stat to allow entering ALPHA commands
		LINK	-Link to another program (perhaps a utility)
		PDUMP	-Dump core or registers on request
		FERROR	-Deactivate error handler
COPY	EVQEND	none	Write WORKIN file on MASPRINT (INPUT) (PRINT)
On error (see Section 5.5.9.3)	EVQERR		Recovers upon error interrupt: first dumps registers, then allows commands, input is requested by DDNAME 'ERRIN'
		PSW	Restart with new PSW
		D	Dump core.
		R	(Actually anything with a blank); reset system interrupt element and return to operating system

Table 5.26 System Programmer's Functions

The operation of the error handler is a straight forward system function. EVQERR is a totally separate CSECT from the rest of the system, this is to prevent any EVQS system error from destroying necessary parts of EVQERR. The initiate request of EVQBGN (BEGIN: ERROR) calls EVQERR which issues a SPIE MACRO; which replaces the operating system interrupt mask with one that reacts on all interrupts and changes the operating system's

reaction, upon a program interrupt, to one of branching to a location within EVQERR (specifically ERRR). When any program interrupt occurs the operating system refers to the program interruption control area (PICA) set to the EVQERR routine by the previous SPIE and passes a program interruption element (PIE) to ERRR -- which then dumps the 'registers at ERROR' from within the PIE and requests further action via the 'ERRIN' DDNAME.

The EVQEND function is supplied to allow simple file lists at debugging time--it is advisable to work with a test file rather then a full data file so such lists are not excessively long. As WORKIN is the input file, only QSAM files may be read, i.e., Files 2 - 5.

Complete, specific definitions of these programs follow.

5.5.9.1 BEGIN Command (System Programmer's Command)

	Form
ERROR calls initializing routine for error/interrupt recovery.	1
LINKS = (NAME,'STR')	
= (NAME)*	3
(note how specially handled)	

LINKS to module named using STR as parameters, passed in normal invocation manner, if present in either STEPLIB or system LINKLIB.

*The LINKS symbolic name and parameters are copied exactly so no blanks are allowed except inside the quotes.

FERROR restores initial (PICA) error exit status — Form

(returns to normal system action). — 1

WAIT = MMSS initiates wait state for MM minutes

and SS seconds; by issuing STIMER MACRO, i.e., — 2

STIMER WAIT, DINTVL=A(DOMMSSOO).

PDUMP = (START ADR-STOP ADR)**

XXXXXX-XXXXXX (Print addresses's contents) — 4

= R (Print all registers) — 2

5.5.9.2 List Command Copy

Simple Listing Command

EVQEND: --'COPY' command

Lists unit assigned as INPUT (2 - 5)

Writes on unit assigned to PRINT

(terminal or printer, 2 - 5)

5.5.9.3 Error Recovery Commands

On error--The following will be dumped:

Old PSW --- 8 bytes,

Registers Ø - 15 -- 4 bytes each.

The input for DDNAME 'ERRIN' will be requested; the possible responses are:

(All start in column 1 and are fixed format--exactly as shown.)

To dump core: DØXXXXXXØXXXXXX

Ending address

Starting address

To restart: PSWØXXXXXX

Restart address

**Both addresses require preceding zeroes.

ERROR RECOVERY LOGIC

CSECT : EVQERR

Must be compiled separately from the rest--

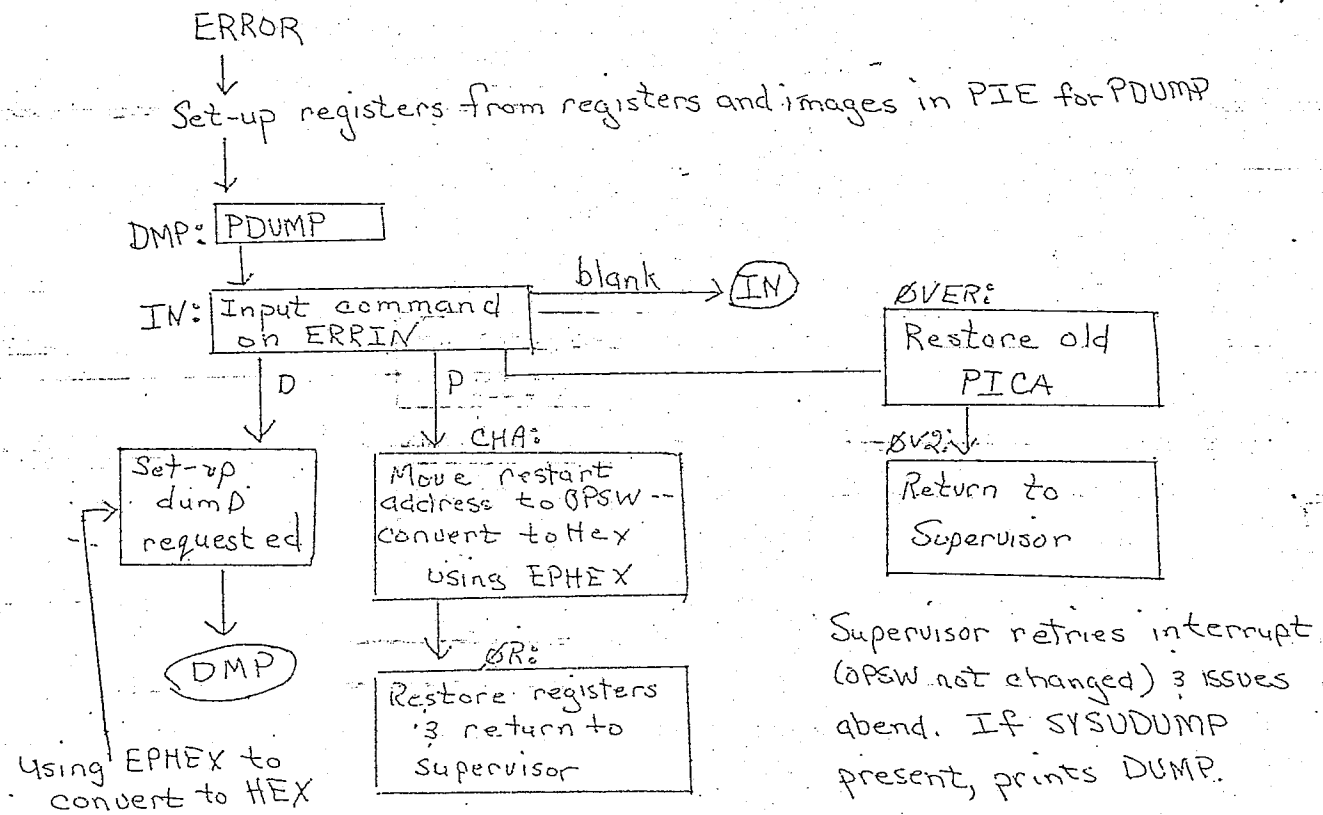
Use ASMG batch option

SAVE

Issue SPIE ERRR, ((1,15))

Return with old PICA address in REG1.

On error OS supervisor goes to:



EPHEX → same as used by EVQBGN Dump function

To terminate: X ---Any character except Ø, P, D, terminates by letting interruption proceed (and it will cause a dump if the required DD card is present).

No effect: Ø

5.5.9.4 LOGIC Notes (on System Programmer's Commands)

5.5.9.4.1 Error Facilities

5.5.9.4.1.1 SPIE Issuing an SPIE MACRO sets the OS response to an error interrupt. The parameter '((1,15))' causes the SPIE error routine to pass control to the user routine named, upon any program interrupt, and changes OS program interruption control area (PICA) to do so. After the SPIE, register 1 has the address of the old PICA which must reset at the end of the program by issuing SPIE MF=(E,(register)) where A(PICA) was loaded into register given.

On An Error Register 1 points to the program interruption element (PIE) which contains: ;and the registers contain:

Bytes		PIE	Register	
Ø	XX	PICA address	1	A(PIE)
4		Old PSW at	2 - 12	Same as program at ERROR
8		interruption	13	ADR of main routine SAVE area
12	Register	14	14	Return address to supervisor
16		15	15	Address of error routine
20		Ø		
24		1		
28		2		

To restart at a new point, change the last six bytes of the old PSW in the PIE. Bits 16 - 31 of OPSW give the error code. Upon return to the system, the PIE register images are reloaded into the registers, so changing the images will effectively change registers Ø - 2, 14 - 15.