KDF-9 Atlas Autocode Compiler

Documentation Vol 3

P-LABELS

1P: routine entry

2P: print time

3P: caption sequence

4P: return

5P: test for valid cycle

6P: stop sequence

7P: switch label

8P: OVR set

9P: non-integral integer quotient

10P: n1/n2/n3

11P: result not specified

12P: read sequence (input) 48P: 49P:

13P: print char (output) 15P: 16P:

14P: div N2 by 10 ↑ N1

15P: mult N2 by 10 ↑ N1

16P: set up for print routines

17P: print N1 digits of N2

18P: used in sequence 17P:

19P: standardies acc

20P: query prints

21P: query prints

22P: LR table

23P:

24P:

25P:

26P:

27P: error in sin

28P:

29P: array inside out

30P: input case /1/ input buffer pointer

31P: rubbish /1/ reconstructed line pointer

32P: Table look up for output symbols 0-32

33P: Table look up for output symbols 64-96

34P: Table look up for lower case characters

35P: bounds of first input buffer

36P: output case /1/ output buffer pointer

37P:

38P: bounds of first output buffer pointer

39P: Table look up for input characters 15-31

40P: reconstructed line pointer

41P: powers of 10

42P: log 2 (base 10)

43P: end of line reconstruct buffer

44P: bounds of second input buffer

45P: bounds of second output buffer pointer

46P: 999: set up at start of compiling and again at start of run of compiled program

47P: print single symbol

48P: line reconstruct

49P: read character

50P: fill PROGRAM block (in compiler)

51P: fill LABEL block (in compiler)

52P: fill SET block (in compiler)

53P: M/C

54P: fill PROGRAM block (in compiler)

55P: M/C

56P: M/C

57P: M/C

58P: end of perm

59P: M/C

60P: M/C

61P: entry sequence

62P: JOB HEAD

63P: M/C

64P: ENTERED and AA FAIL

65P: masks

66P: (compute array element address)

67P: compute allocation & control vector

68P: dump control vectors

69P: array control vector

70P: SUM SERIES, FIXED POINT, SHIFTED (for mathematical routines)

71P: constants for sin,cos

72P: constants for tan

73P: constants for log

74P: constants for exp

75P: constants in arctan

76P: constants in arctan

77P: constants in arctan

78P: constants in arctan

79P: constants in arctan

80P:

81P: exponent in real expression

82P: exponent in integer expression

83P:

84P:

85P: fault trapping vector

86P: print char in N2 N1 times

87P: print character and store output buffer pointer

88P: output buffer pointer to Q15

89P: Parameter and tape identifiers in job heading

90P: monitor sequence

91P: Elapsed time from midnight (starting time)

Computation time from midnight

92P: Title line, stored six characters to a word

93P: Radix word for Hours, Mins, Secs

94P: Date compiler defined

95P: excess blocks

96P:

97P: print character

98P: initial output for line printer

99P: initial output for paper tape (or mag)

P-LABELS, VER-I

1P:		589
2P:	3/	590
3P:		623
4P:	4/	632
5P:	1/	676
6P:	4/	681
7P:	4/	727
8P:	5/	736
9P:	2/	740
10P:	5/	742
11P:	4/	744
12P:	1/	747
13P:	1/	835
14P:	4/	874
15P:	3/	879
16P:	5/	884
17P:	4/	893
18P:	2/	895
19P:	3/	898
20P:	2/	910
21P:	3/	919
22P:		0
23P:		0
24P:		0
25P:		0
26P:		0
27P:	2/	1531
28P:		0
29P:		931
30P:		226
31P:		227
32P:		435
33P:		468
34P:		501
35P:		192
36P:		228
37P:		0
38P:		384
39P:		518
40P:		229

41P:		153
42P:		159
43P:		350
44P:		224
45P:		416
46P:	3/	948
47P:	1/	1019
48P:	4/	1065
49P:	2/	1177
50P:		0
51P:		0
52P:		0
53P:		0
54P:		0
55P:		0
56P:		0
57P:		0
58P:		1945
59P:		0
60P:		0
61P:	3/	1865
62P:		0
63P:		0
64P:		1882
65P:		1880
66P:		1201
67P:		1219
68P:	2/	1231
69P:		1207
70P:	3/	1518
71P:		1534
72P:		1602
73P:		1676
74P:		1703
75P:		1775
76P:		1778
77P:		1779
78P:		1785
79P:		1780
80P:		0

81P:	3/	1241
82P:	5/	1249
83P:		0
84P:		0
85P:		634
86P:		1363
87P:	2/	1361
88P:	2/	1359
89P:		550
90P:		666
91P:		536
92P:		538
93P:		535
94P:		549
95P:	3/	933
96P:		568
97P:	3/	836
98P:	1/	988
99P:	4/	978

P-LABELS, VER-J

1P:		506
2P:	3/	507
3P:		540
4P:	4/	551
5P:	1/	595
6P:	4/	600
7P:	2/	645
8P:	3/	654
9P:		658
10P:	3/	660
11P:	2/	662
12P:	5/	664
13P:	1/	752
14P:	4/	789
15P:	3/	794
16P:	5/	799
17P:		809
18P:	4/	810
19P:	5/	813
20P:	4/	825
21P:	5/	834
22P:		0
23P:		0
24P:		0
25P:		0
26P:		0
27P:	2/	1374
28P:		0
29P:	2/	846
30P:		226
31P:		227
32P:		435
33P:		0
34P:		0
35P:		192
36P:		228
37P:		0
38P:		384
39P:		0
40P:		229

41P:		153
42P:		159
43P:		350
44P:		224
45P:		416
46P:	5/	863
47P:	1/	937
48P:	4/	979
49P:	4/	1047
50P:		0
51P:		0
52P:		0
53P:		0
54P:		0
55P:		0
56P:		0
57P:		0
58P:		1788
59P:		0
60P:		0
61P:	3/	1708
62P:		0
63P:		0
64P:		1725
65P:		1723
66P:	2/	1069
67P:		1087
68P:	2/	1099
69P:		1075
70P:	5/	1361
71P:		1377
72P:		1445
73P:		1519
74P:		1546
75P:		1618
76P:		1621
77P:		1622
78P:		1628
79P:		1623
80P:		0
81P:	3/	1109

82P:	5/	1117
83P:		0
84P:		0
85P:		553
86P:	1/	1224
87P:	3/	1222
88P:	3/	1220
89P:		467
90P:		585
91P:		453
92P:		455
93P:		452
94P:		466
95P:	5/	848
96P:		485
97P:	2/	753
98P:	5/	903
99P:		894

ON-LINE MESSAGES

PAPER TAPE LOADER FOR PROGRAMS PUNCHED BY ATLAS

AA OFF	CHECKSUM FAILURE	ENDS 0
AA ON	PROGRAM ENTERED	ENDS 2

COMPILER CALL PROGRAM

AA PAR	MAGNETIC TAPE PARITY FAILURE	ENDS 0
AA SUM	CHECKSUM FAILURE	ENDS 0

COMPILER LOADING COMPILED PROGRAM

AA FAIL	CHECKSUM FAILURE	ENDS 0

Locations used by Ver. I

E 418 : Compiler tape device number.
419 : Work tape device number.
420 : Input device number (for paper or mag. tape).
421 : Output device number (for paper or mag. tape).
422 : Output device number (for line printer).
423 : Flag = { -1 : only punch available. / 0 : punch and L.P available. / 1 : only L.P. available.

[mag. tape not distinguished from paper tape after allocation]

424 : Jump instruction to Odd restart.
425 : Jump instruction to Even restart.
426 : Termination time (absolute).
427 : Termination time (computing).
428 & 429 : TIME OK.
430 & 431 : RELOAD.

432
433
434
435 } Slow like
436
437 }

438 Card R. dev. No = (-1) No reader available
439 = 1 if Cards current input = 0 otherwise
440 2nd Reader No.
441 OTHER COMP D. No.

Initialisation sequence 46P:999

Dump computing time to date and time from midnight. Dump time limit and exit if about to enter compiled program. If about to compile dump time limit for compiling, reset input device number and fill both input buffers and set input buffer pointer then look for ***A at head of reconstructed line buffer. On finding it exit with reconstructed line pointer already set.

COMPILER DEFINITION AND ENTRY SEQUENCES

Before the compiler is stored on a magnetic tape a call block is dumped there by a Usercode program called PTFC. This block is used for copying the compiler from the magnetic tape into the store and when it is put in the store to be executed it occupies E8-E100.

The define compiler sequence is in two distinct parts. There is an initialisation sequence which is executed every time the compiler is entered and a definition sequence used only to define the compiler onto a magnetic tape. The second sequence performs among others the following actions. The number of items in the nest is stored in E101, this should be 0, and the contents of the nest are then stored in the succeeding locations. The SJNS is treated similarly starting with E118 which should in this case be 1 and then E119 contains the return address to the define compiler sequence which is entered every time the compiler runs. The contents of the Q-stores are then put in E135 onwards starting with Q1. After this the store between E101 and the current position of the workspace pointer is dumped on the magnetic tape as one block.

A run of the compiler is started by running the Usercode program AASR/I which queries the operator via the monitor typewriter about the devices required and stores the device numbers, reads the call block from the compiler tape into E8-E100 and enters it at 3/30.

This block outputs to the monitor typewriter PROGRAM hh.mm.ss and reads the compiler into store starting at E101. Before exiting via the return address of the J870 instruction in the define compiler sequence it resets the Q-stores and nests, and stores the time at E427.

Subsequently when the stop sequence is entered at the end of compilation of a faulty program or at the end of the run of a compiled program the procedure is as follows. After completion of the output the compiler tape is picked up, the call block found and copied into E8-E100 and entered at 3/30 as above to copy the rest of the compiler into store and continue as before.

CONTENTS OF PERM

Example of Calculation of base and Multipliers

A (a:b, c:d, e:f)

Dump	n	m1	m2	m0	all

Where n = 3

m1 = m2* (d - c + 1)

m2 = f - e + 1

m0 = - (e + a *(d - c + 1)*(f - e + 1) + c * (f - e + 1)) + 1 + n

all = (b - a + 1) * (d - c + 1) * (f - e + 1)

ARRAY ELEMENT ADDRESSES

For arrays of dimension greater than one, on declaration enter at 67P: with dimension in N1 to compute allocation and control vector which is stored at 69P:. Then enter at 68P: to dump it at WS onwards. No vector is dumped for two-dimensional arrays all the information going into the word corresponding to the name, in the multi-dimensional case this contains the work-space pointer prior to dumping the vector. On exit from 68P: also the workspace pointer is moved up an amount equal to the calculated allocation.

For multi-dimensional arrays the address of an element is calculated as m(0)+k+j*m(n-1)+...+i*m(1) where m(0) is correction term and the indices are k to i reading from right to left. The m(i) are the multipliers stored in the control vector.

For two-dimensional arrays the information word is picked up and dumped in Q13. The address calculated is (i*I13)+j+M13 where M13 contains the base address, and I13 is m(1).

```
                comment line reconstruct
48:48P:         *Q14            } save Q-stores to be used
                *Q15            }
                begin
                integer char
                switch h(0:14)
                *SET 30P:       }
                *=M13           } set Q14 to pick up input
                *MOM13          }
                *=Q14           }
                *SET 40P:       }
                *=RM15          } set Q 15 to point to rlb
                *M15            }
                *NEG            } -∝ furthest character /1/∝ current visible char
                *=C15           }
511:            *M+I15          }
520:            *M+I14
                *M14
                *SET B 37
                *AND
                *J 515=Z        -> 515 if necessary to switch buffers
527:            *MOM14          fetch next character
                *DUP            }
                *BITS           } check parity of charicter
                *SHC-1          }
                *J514<Z         -> 514 if parity is odd
                *ERASE
                *SET 14422      ; comment p
                *J513 C11Z                      fault if parity even
                *SET 18
                *JS 90P
                caption # PARITY # FAULT # IN # DATA    fault stop if untrapped
                stop
515:            *MOM14          fetch input parameters into Q13
                *=Q13
                *PICQ13         read thirty characters
                *MOM14N         fetch input buffer base address
                                        into M14
                *=M14
                *J527
```

Notes: rlb = reconstructed line buffer

30P contains input case / 1 / input buffer pointer

40P ∝ rlb

```
514:    *SETB 77
        *AND                mask out -parity bit
        *DUP
        *SET+14     }
        *-          }  -> 512 unless char is space, newline, tab,
        *J512>Z     }                 case shift, stop code, blank tape...
        **=char
        ->h(char)   switch to special character handeling
h(0):   *M15        for blank tape
        *C15
        *+
        *J511<Z     -> 511 if rlb empty (ignore the blank)
        *M15
        *SET 43P:
        *-
        *J526 >Z    -> 526 if rlb full
        *DC15
        *SET+65
        *=MOM15
        *J511
h(1):h(8):h(10):h(11):h(13):h(14):*SET 15062   ; comment u  for illegal codes
        *J 513 C11 Z
        *SET 19                                          stop unless
        *JS 90P                                            trapped
caption # UNASSIGNED # CHAR # IN # DATA # = ; print (char,1,0)
        stop
538:    *ERASE
539:    *ERASE                                      blank
526:    *J510 C11 NZ    comes here if trying to add to full rlb
        *SET 40P:       error stop at 510 unless it is accidental
        *=RM15
                        reset rlb pointers
        *M+I15
        *M15
        *NEG
        *=C15
        *SET 14038          ; l m
        *=MOM15
        *J511           -> 511 to pick up next character
510:    *SET 20
        *JS 90P
        caption # MORE # THAN # 120 # CHARS # ON # LINE
        stop
```

```
h(4):   *M15            tab
        /*SET 40P:
        *DUP
        *PERM
        *-              fetch relative position from beginning of rlb
        *SHL-3          remove least significant octal digit. (if it is > 3,
        *DUP                    then a tab will be a multiple of 16)
        *SET 3
        *-
        *J 517 ≤Z       -> 517 for 8 space tab
        *I15            set tab length to 16 spaces
        *OR
        *SET 15
        *J 538=         -> 538 if rlb will overflow
517:    *NOT
        *NEG
        *SHL+3
        *+              compute new visable character posn
        *C15
        *NEG
        *DUP
        *=M15
        *REV
        *DUP
        *PERM
        *-
        *J528≥Z
529:    *M+I15
        *SET+65         plant spaces to new posn.
        /*=MOM15
        *M15
        *J529≠
        *NEG
        *=C15
        *J511
528:    *=M15
        *J511
h(5):   *M-I15          for backspace, decriment current visible ptr.
        *M15
        /*SET 40P:
        *-              check whether ptr off line left
        *J520≠Z
        *M+I15          increment if so
        *J520
```

Note: first five tabs are multiples of 8: 0, 8, 16, 24, 32

rest are multiples of 16: 48, 64, 80, 96, 112

```
h(6):   *CO TO Q14      for lower case, change case to lower
        *J520
h(7):   *I12            for upper case, change case
        *=C14
        *J520
h(9):   *SET 73         for color shift, set internal code
        *J513           →513
h(12):  *SET 76         For stop code, set internal code
        *J513           →513
512:    *C14            fetch input case    here for non-special char
        *J516=Z         → 516 if lower case
        *DUP
        *SET+26         → 513 if / or digit
        *-
        *J513<Z
        *DUP
        *SET+28
        *-
        *J530≤Z         → 530 if α or β
        *SET+63
        *J513≠          → 513 unless char is erase
530:    *SET+64
        *+              } compute internal code for α, β, or erase
        *J513           }          →513        ie., 90, 91, or 127
516:    *SET+64                                come here if lower case
        *+              add 64 to code.
        *DUP
        *SET+95
        *-
        *J513>Z         → 513 if char is letter
        *SET+79
        *-
        *SET 39P:       } compute address of translation of char in
        *+              } dictionary  (15 ≤ char ≤ 31)
        *=M13
        *MOM13          fetch translation
```

```
                                comes here to plant character
513:    *M15
        *C15
        *+
        *J518≤Z         → 518 if char to be compounded
        *M15
        *SET 43P:
        *-
        *J539≥Z         → 519 if more than 120 chars already planted
        *DC15
519:    *=MOM15         plant single character
        *J511
518:    *SET+127
        *J519=          → 519 if char is erase
        *MOM15          fetch part to be compounded (= part 1)
        *SET+65
        *J531≠          → 531 if not blank
        *ERASE
        *=MOM15         plant char
        *J511
531:    *SET+127
        *J532≠          → 532 if part 1 not erase
533:    *ERASE
        *ERASE
        *J511           plant char
532:    *DUP
        *=Q13
        *SETB 177
        *AND
        *DUP
        *PERM
        *J533=          → 533 if part 1 = char (overprinting)
        *REV
        *Q13
        *J521≠          → 521 if ~~early~~ part 1 already compound
        *DUP D
        *-
        *J522<Z
        *REV
```

```
522:    *SHL+7
        *+
        *=MOM15
        *J511
521:    *REV
        *Q13
        *SHL+34
        *SHL-41
        *DUP
        *PERM
        *J534≠
535:    *ERASE
        *ERASE
        *ERASE
        *J511
534:    *Q13
        *SHL-14
        *DUP
        *J523≠Z
        *ERASE
        *CAB
        *DUP D
        *-
        *J524>Z
        *ERASE
        *Q13
        *SHL+7
        *+
        *=MOM15
        *ERASE
        *J511
```

compound char and part 1

→ 534 if char ≠ component of part 1

→ 523 if ~~three chars~~ part 1 has 3 components

→ 524 if code for 1st component > code for char

compound and plant

```
524:    *PERM
        *DUP D
        *-
        *J525<Z          → 525 if char > 2nd component
        *REV
        *SHL+7
        *+
        *SHL+7           new triple = ((2nd *2^7) + char)*2^7 + 1st
        *+
        *=MOM15
        *J511
525:    *SHL+14
        *Q13
        *+
        *=MOM15          new triple = char*2^14 + part 1
        *ERASE
        *ERASE
        *J511
523:    *J535=           → 535 if char = 3ed component
        *ERASE
        *ERASE
        *ERASE
        *SET 14806               ; 1.s
        *J 513 C11 Z
        *ERASE
        *SET 15
        *JS 90P
        caption ∧ MORE ∯ THAN ∯ THREE ∯ SYMBOLS ∯ IN ∯ ONE ∯ PRINTED ∯ POSITION
        stop
536:    *ERASE           ignore insignificant space or erase
        *M-I13
        *J 537
```

```
h(2):h(3):*C15
          *NEG
          *=RM13
537:      *MOM13
          *SET+65
          *J 536=
          *SET 127
          *J 536=
          *ERASE
          *M+I13
          *SET+4
          *=MOM13
          *M+I13
          *ZERO
          *NOT
          *=MOM13
          *SET 30P:
          *=M13
          *Q14
          *=MOM13
          *SET 31P:
          *=M15
          *SET 40P:
          *=RM13
          *Q13
          *=MOM15
          end
          *=Q15
          *=Q14
          *EXIT 1
```

for newline or page throw

Q13 = 0/1/Furthest visual char

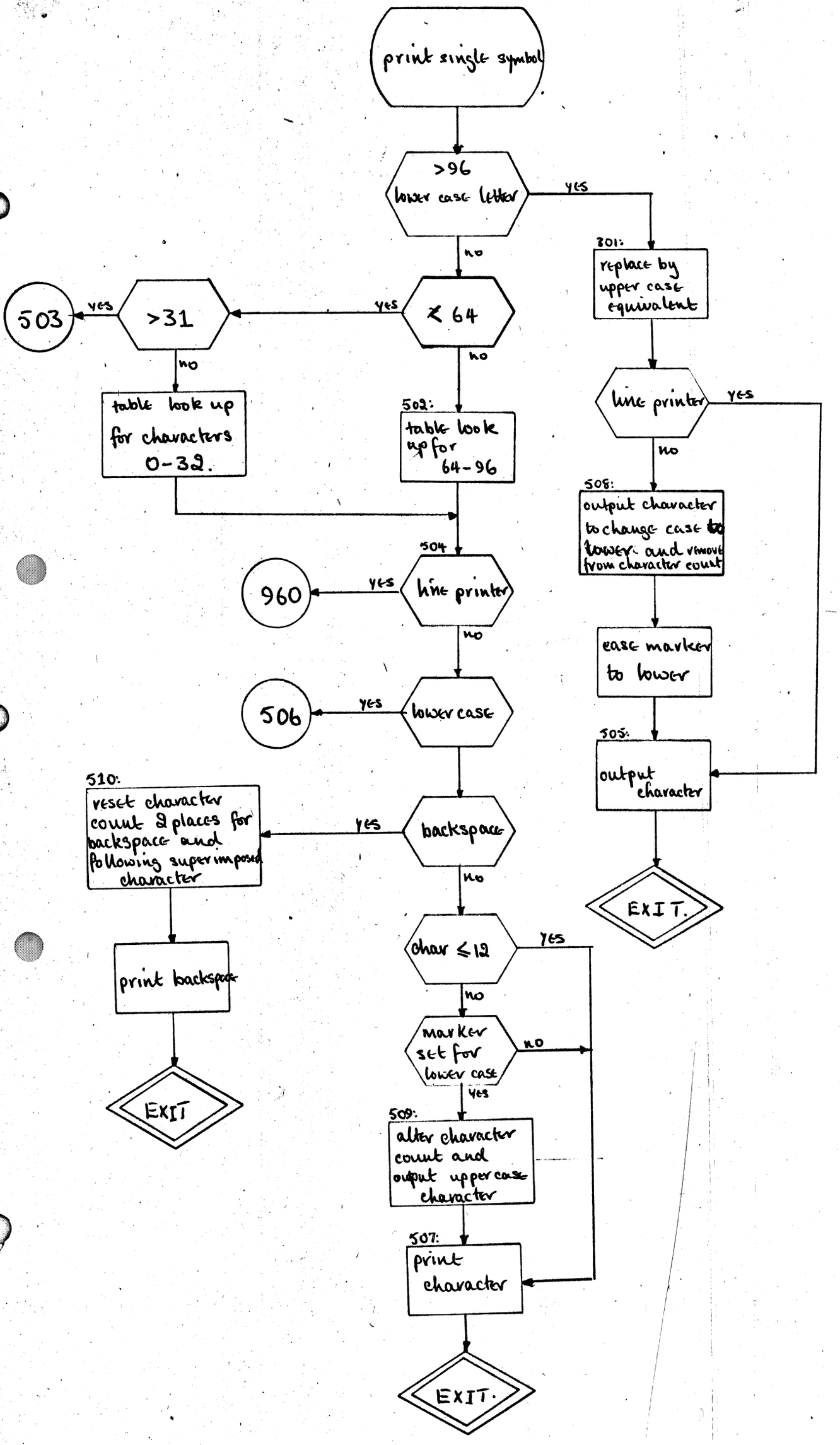
ignore spaces & erases at end of line

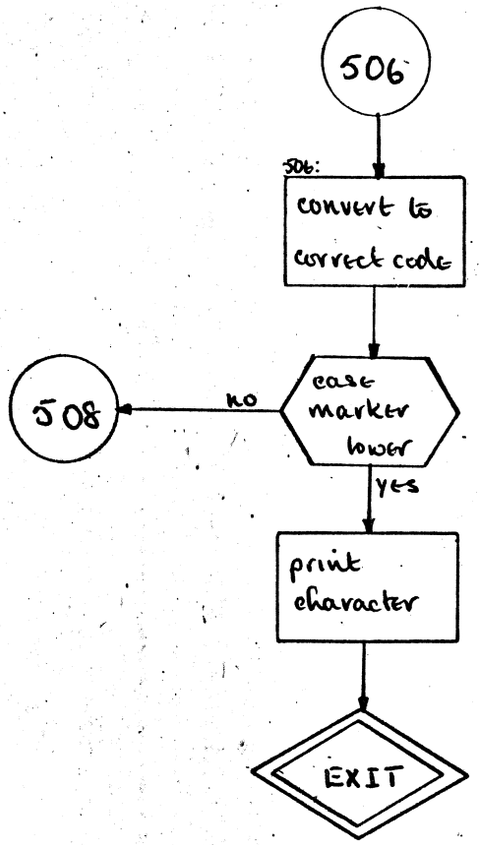
plant newline

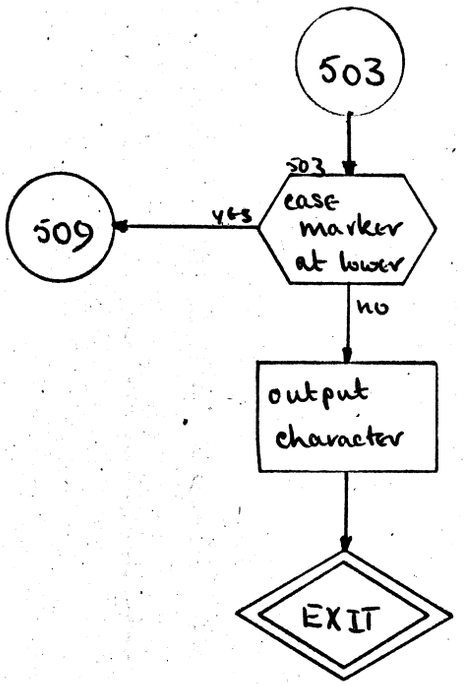
plant -1

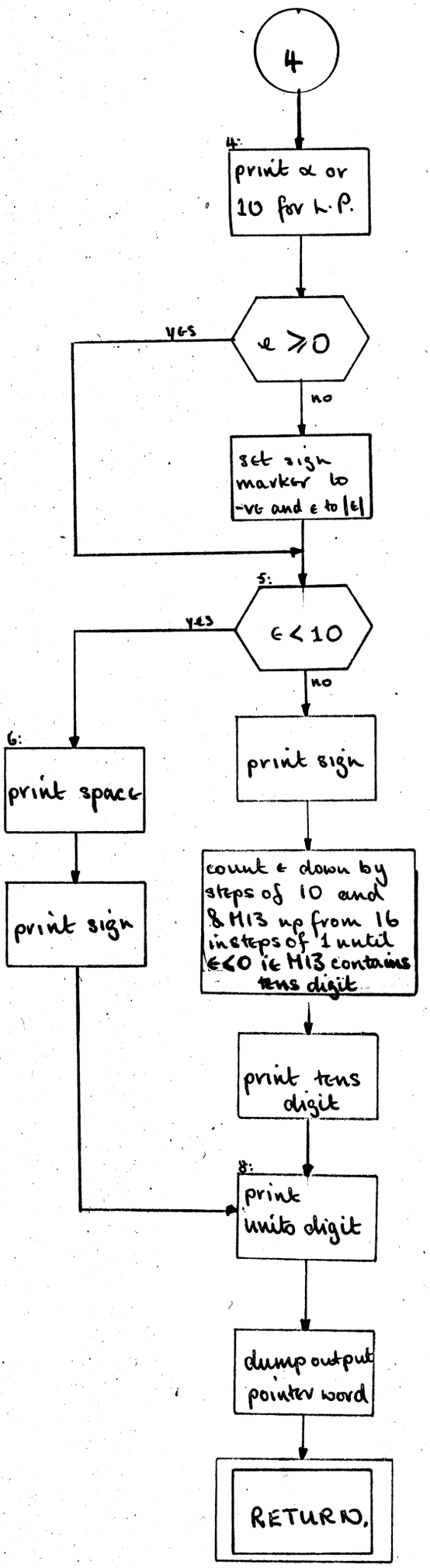
reset Q13

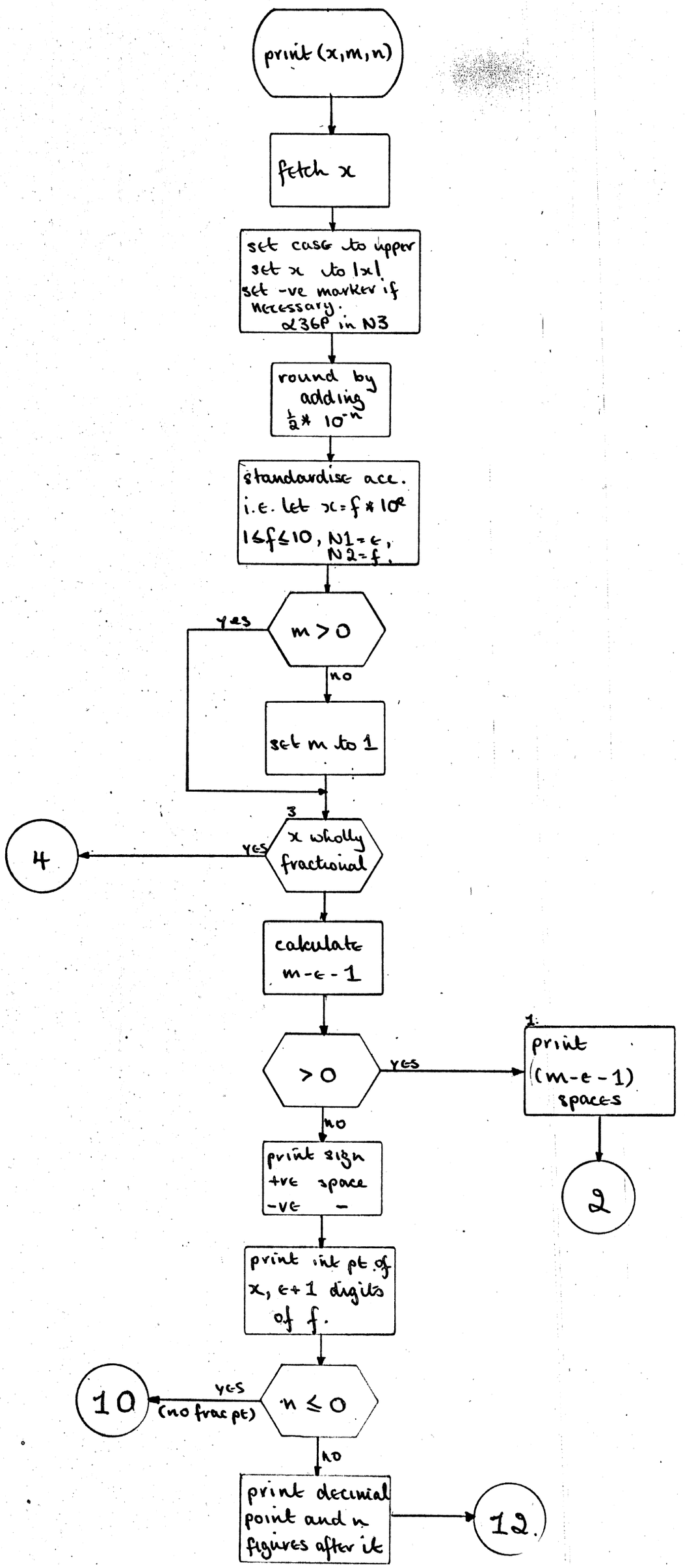
dump input & rtb pointers

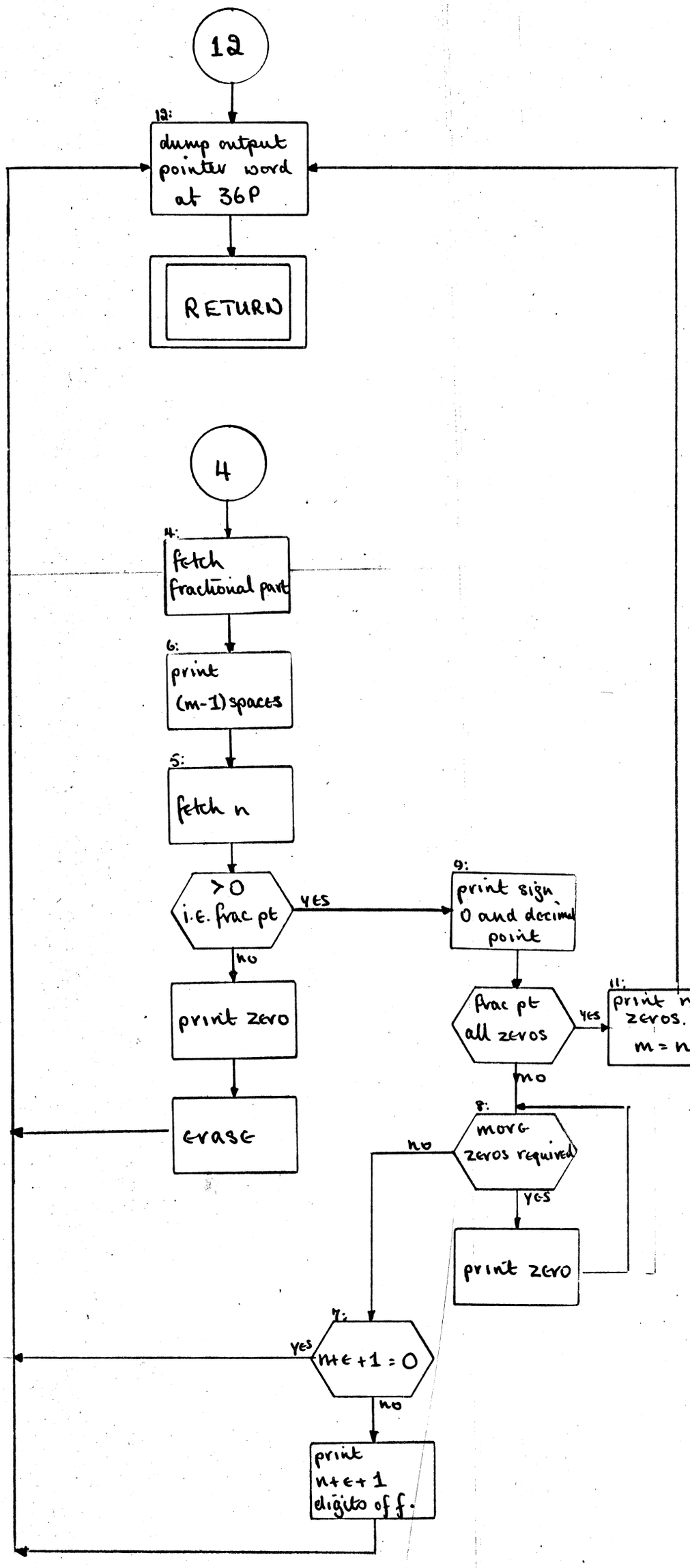

print single symbol
>96 lower case letter
yes
no
301:
replace by upper case equivalent
line printer
yes
no
508:
output character to change case to lower and remove from character count
case marker to lower
505:
output character
EXIT.
< 64
yes
no
503
yes
>31
no
table look up for characters 0-32.
502:
table look up for 64-96
504
960
yes
line printer
no
506
yes
lower case
510:
reset character count 2 places for backspace and following superimposed character
yes
backspace
no
print backspace
EXIT
char ≤ 12
yes
no
marker set for lower case
no
yes
509:
alter character count and output upper case character
507:
print character
EXIT.

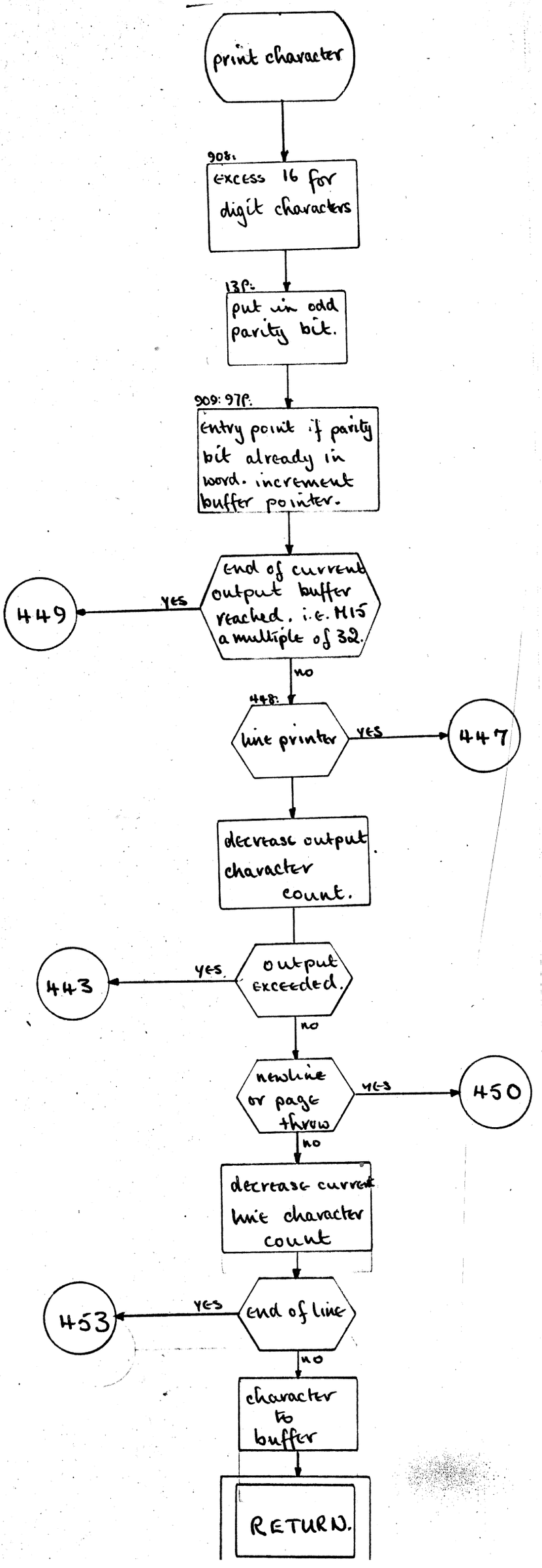

506
506:
convert to
correct code
case
marker
lower
no
508
yes
print
character
EXIT

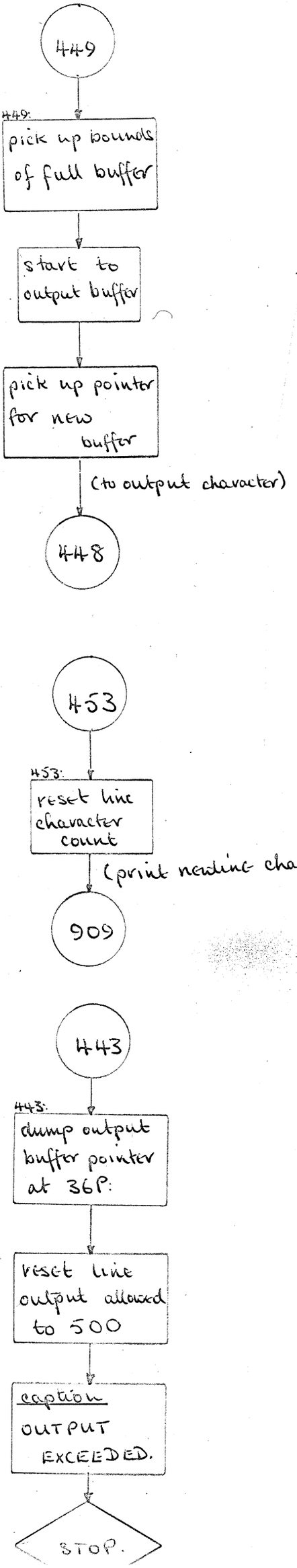

503
503
case
marker
at lower
yes
509
no
output
character
EXIT

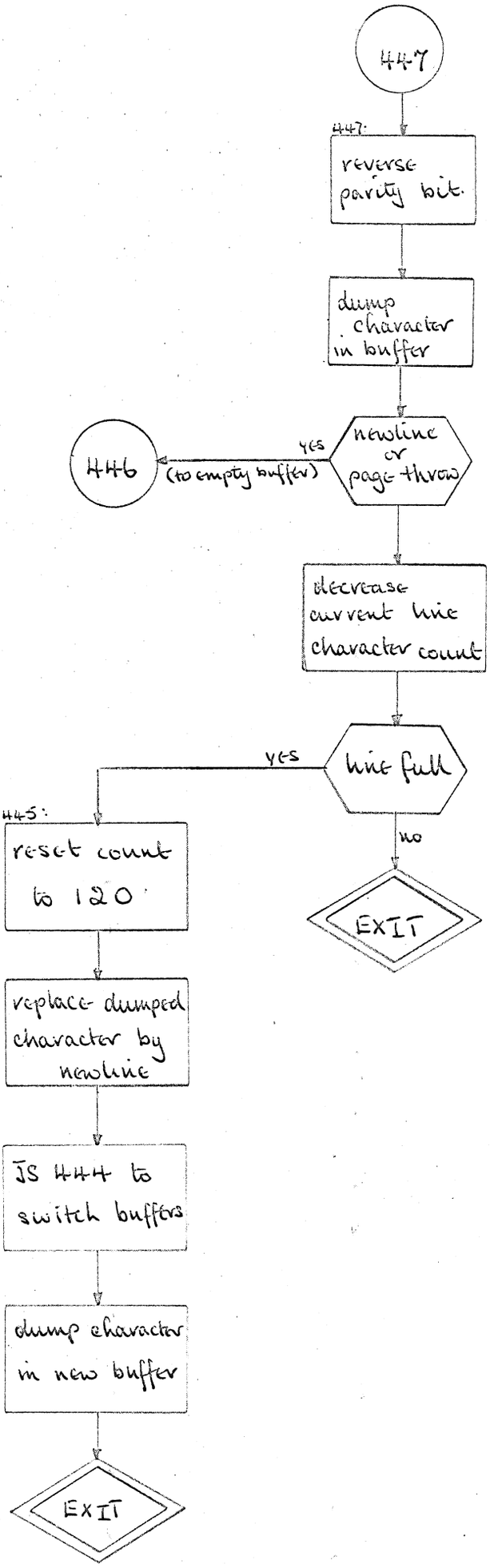

4
4:
print α or 10 for L.P.
e ⩾ 0
yes
no
set sign marker to -ve and ε to |ε|
5:
ε < 10
yes
no
6:
print space
print sign
print sign
count ε down by steps of 10 and &H13 up from 16 in steps of 1 until ε<0 i.e. H13 contains tens digit
print tens digit
8:
print units digit
dump output pointer word
RETURN.

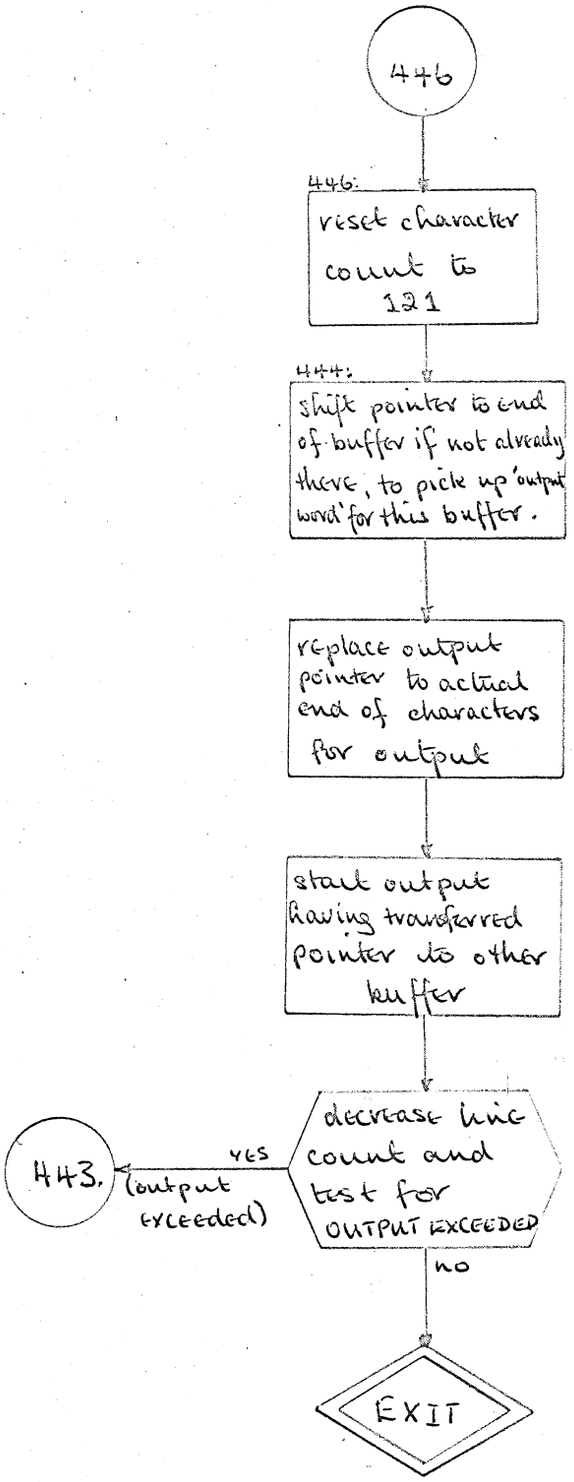

print (x,m,n)
fetch x
set case to upper
set x to |x|
set -ve marker if necessary.
ALGOL in N3
round by adding
½ * 10^-n
standardise acc.
i.e. let x = f * 10^e
1 ≤ f ≤ 10, N1 = e,
N2 = f.
m > 0
yes
no
set m to 1
3
x wholly fractional
yes
4
calculate m - e - 1
> 0
yes
1
print (m-e-1) spaces
2
no
print sign
+ve space
-ve -
print int pt of x, e+1 digits of f.
n ≤ 0
yes
(no frac pt)
10
no
print decimal point and n figures after it
12.

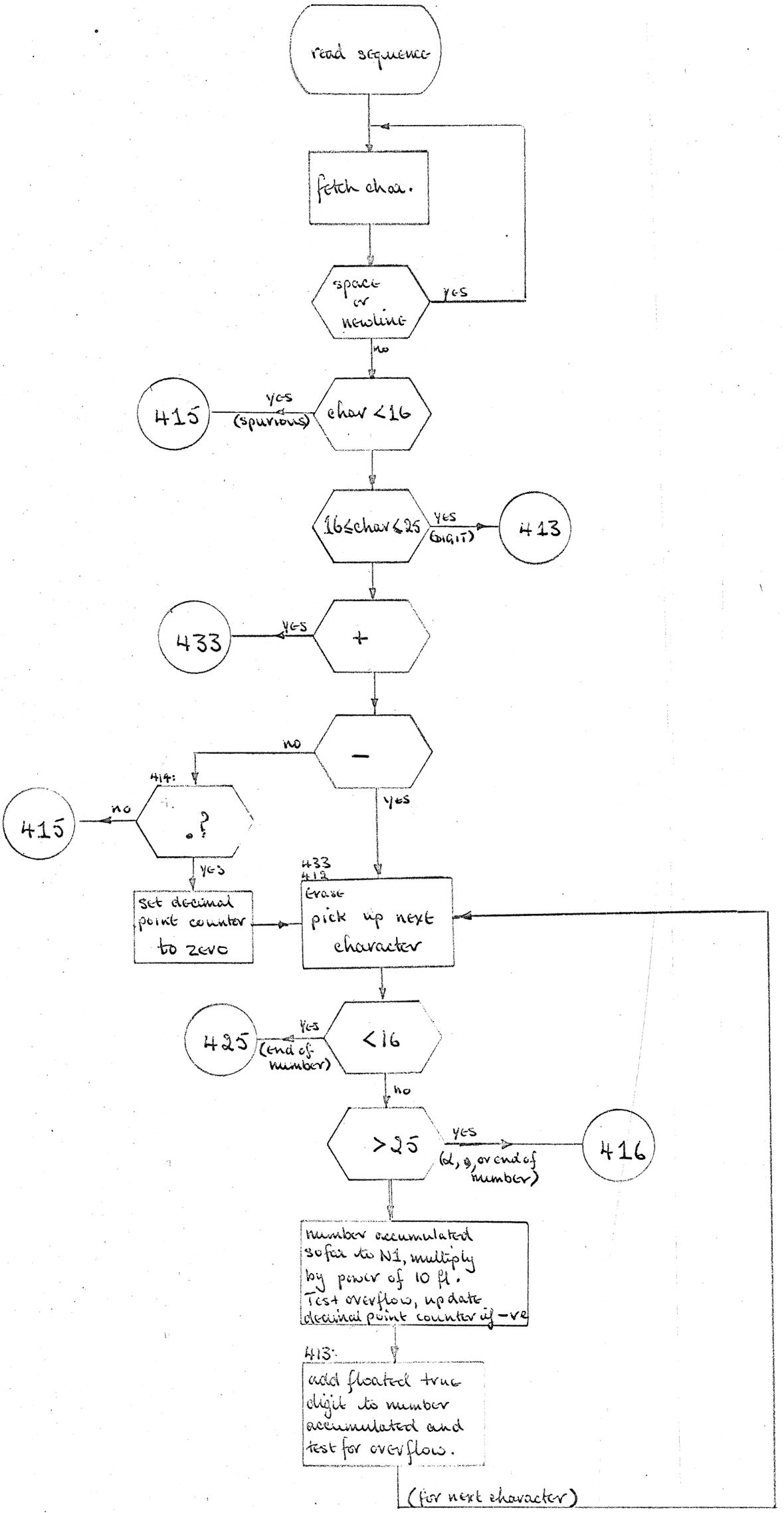

12
13:
dump output pointer word at 36P
RETURN
4
4:
fetch fractional part
6:
print (m-1) spaces
5:
fetch n
>0 i.e. frac pt
yes
no
9:
print sign, 0 and decimal point
print zero
erase
frac pt all zeros
yes
11:
print n zeros. m = n.
no
8:
more zeros required
no
yes
print zero
7:
n+e+1 = 0
yes
no
print n+e+1 digits of f.

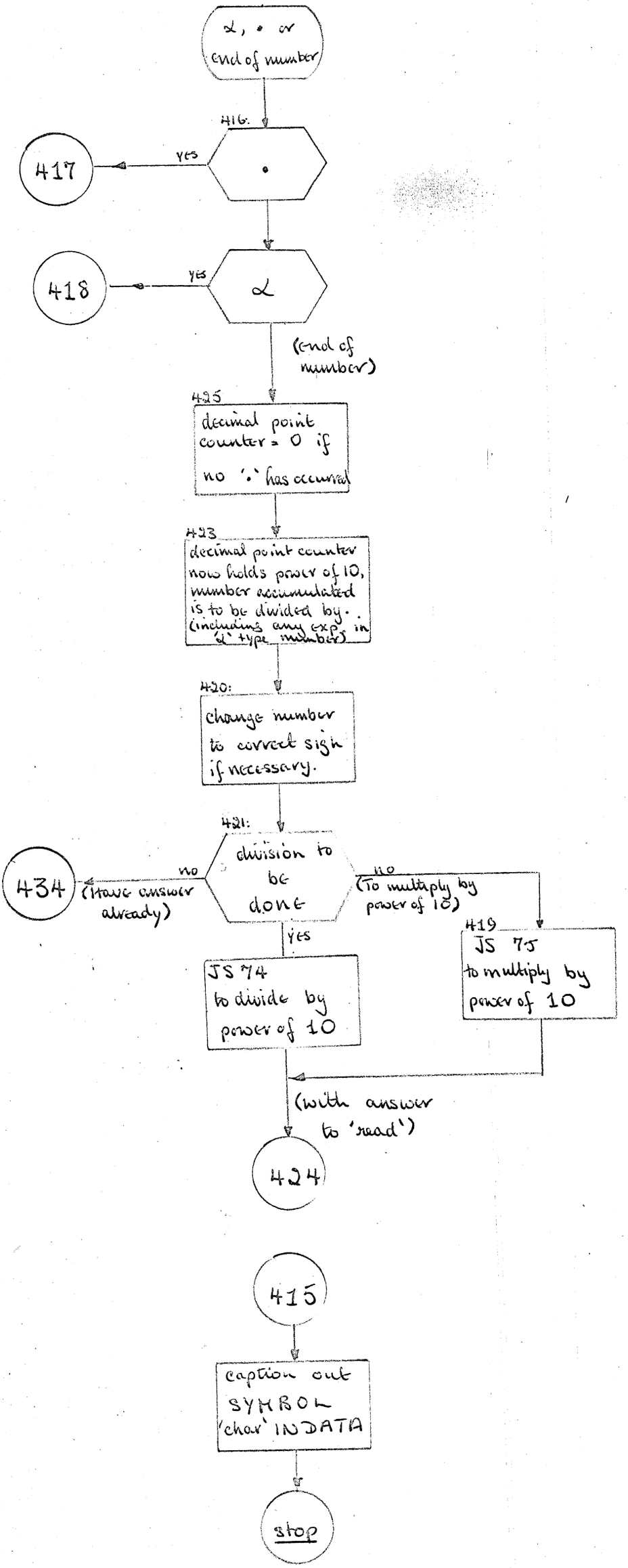

print character
908:
excess 16 for digit characters
13F:
put in odd parity bit.
909: 97F:
entry point if parity bit already in word. increment buffer pointer.
End of current output buffer reached. i.e. M15 a multiple of 32.
yes
449
no
448:
line printer
yes
447
decrease output character count.
output exceeded.
yes
443
no
newline or page throw
yes
450
no
decrease current line character count
End of line
yes
453
no
character to buffer
RETURN.

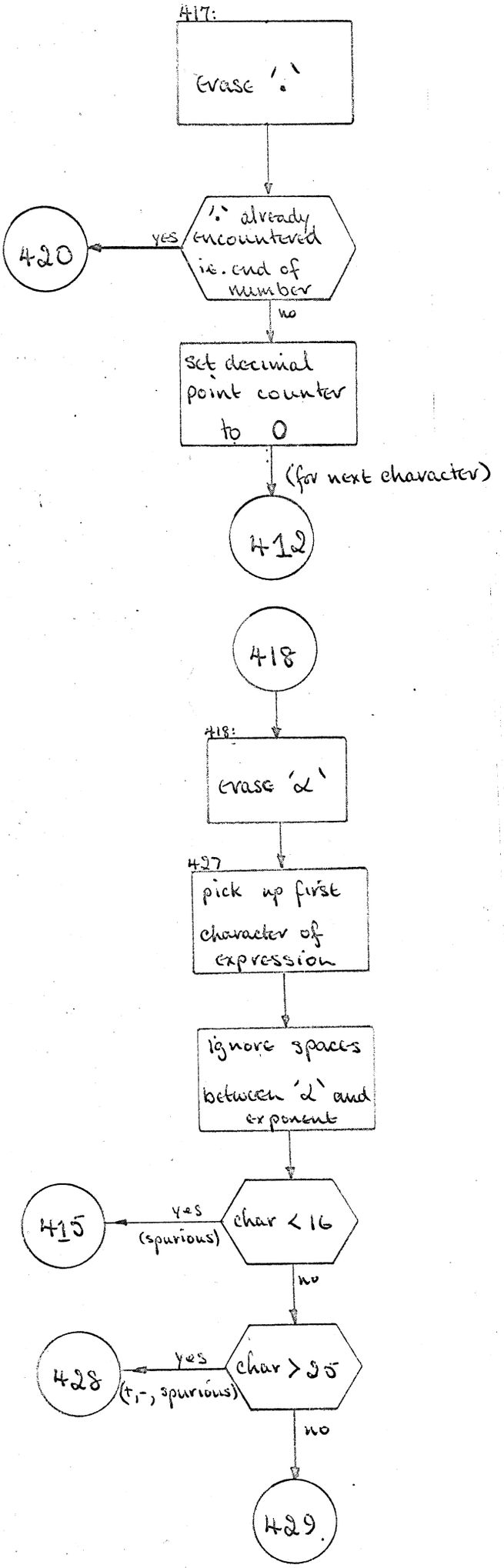

449
449:
pick up bounds of full buffer
start to output buffer
pick up pointer for new buffer
(to output character)
448
453
453:
reset line character count
(print newline character)
909
443
443:
dump output buffer pointer at 36P:
reset line output allowed to 500
caption
OUTPUT EXCEEDED.
STOP.

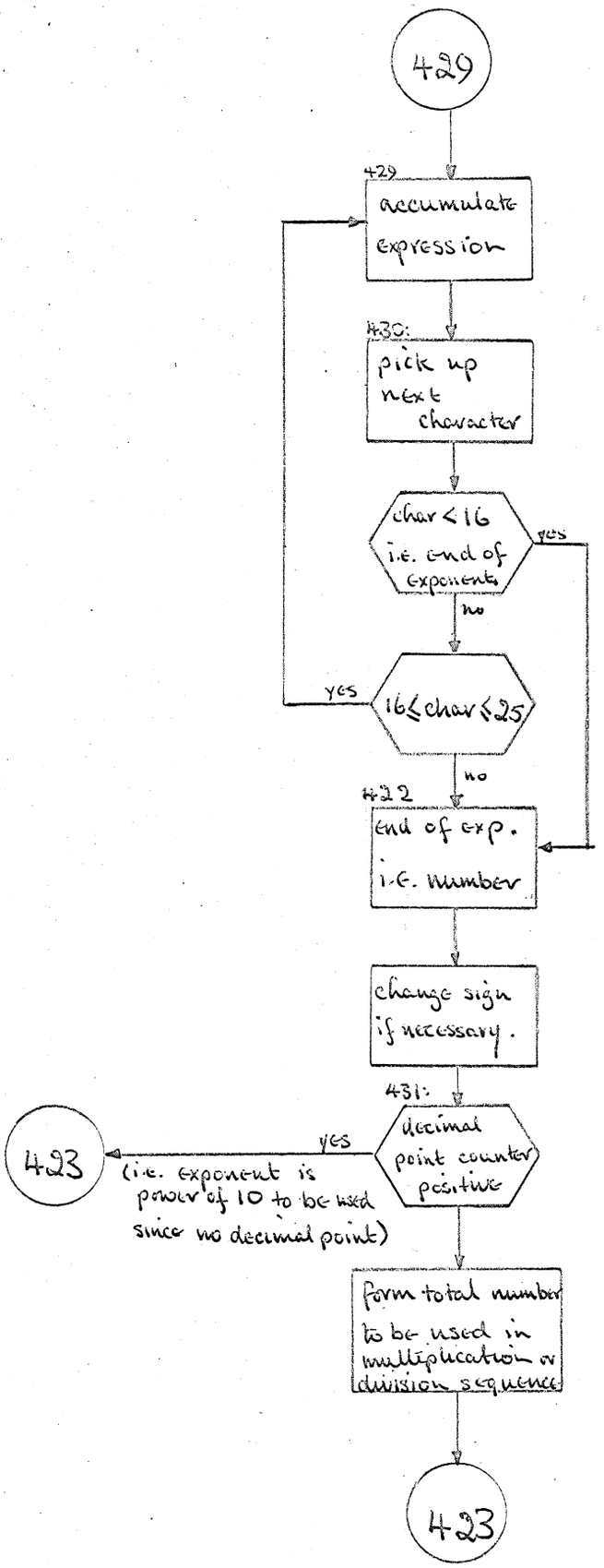

447
447:
reverse parity bit.
dump character in buffer
newline or page throw
yes
446
(to empty buffer)
decrease current line character count
line full
yes
no
445:
reset count to 120
EXIT
replace dumped character by newline
JS 444 to switch buffers
dump character in new buffer
EXIT

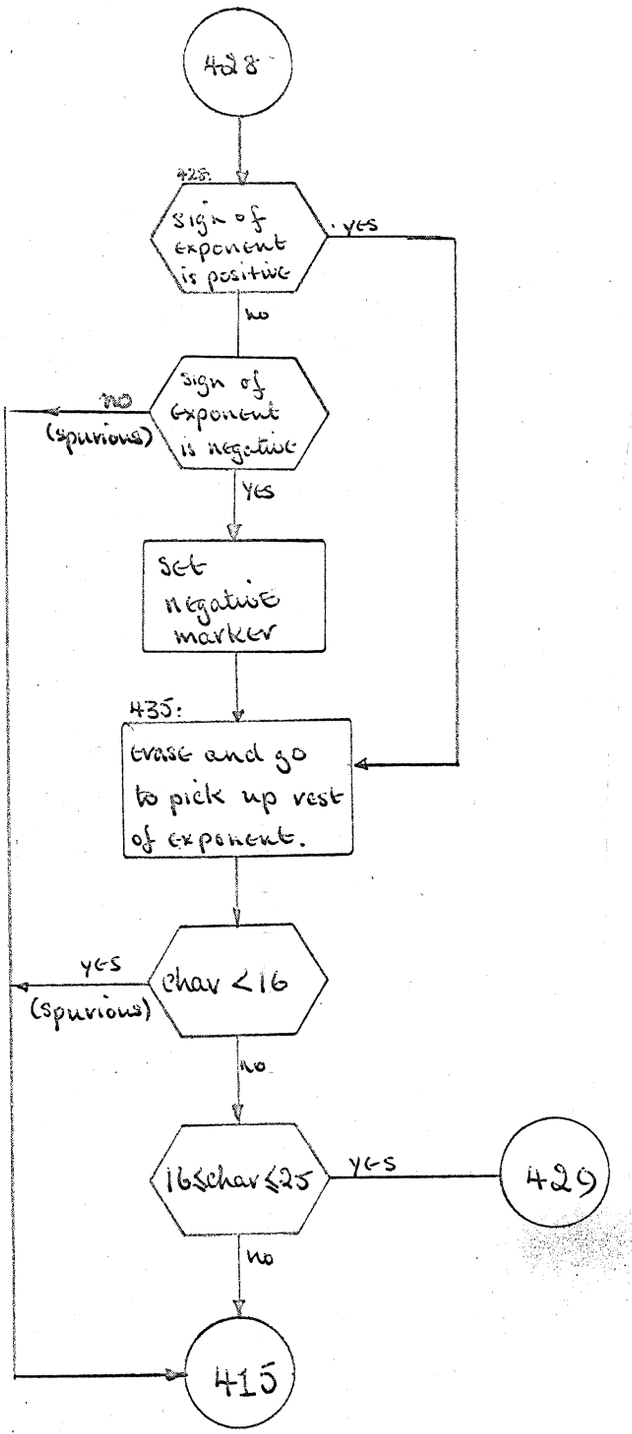

446
446:
reset character count to 121
444:
shift pointer to end of buffer if not already there, to pick up 'output word' for this buffer.
replace output pointer to actual end of characters for output
start output having transferred pointer to other buffer
decrease line count and test for OUTPUT EXCEEDED
YES
(output exceeded)
443.
no
EXIT

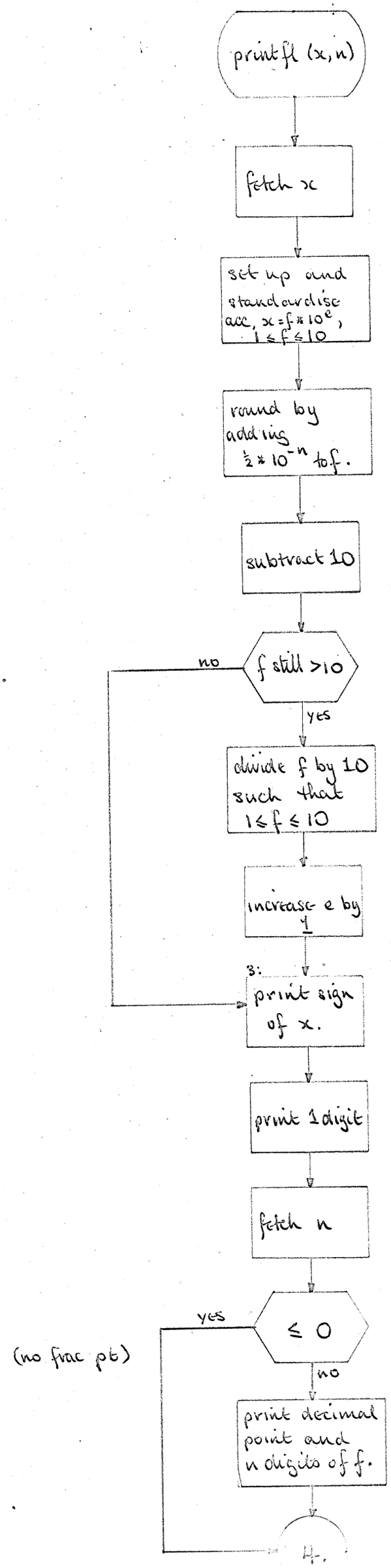

read sequence
fetch char.
space or newline
yes
no
415
yes (spurious)
char < 16
16 ≤ char ≤ 25
yes (digit)
413
433
yes
+
no
−
414:
.?
no
yes
set decimal point counter to zero
433 419
erase
pick up next character
425
yes (end of number)
< 16
no
> 25
yes (L, S, or end of number)
416
number accumulated so far to N1, multiply by power of 10 fl. Test overflow, update decimal point counter if −ve
413:
add floated true digit to number accumulated and test for overflow.
(for next character)

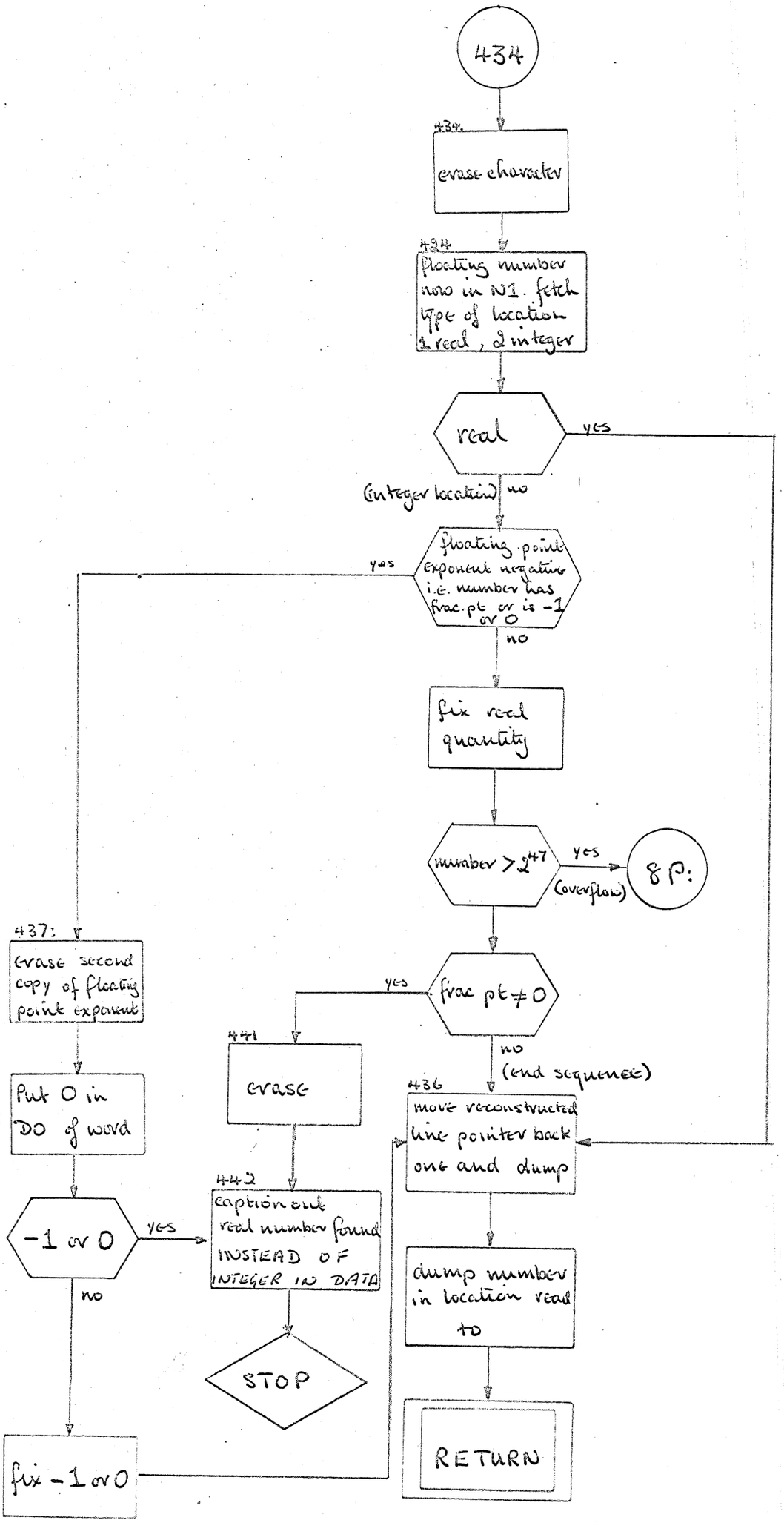

d, . or end of number
416:
417
yes
.
418
yes
d
(end of number)
425
decimal point counter = 0 if no '.' has occurred
423
decimal point counter now holds power of 10, number accumulated is to be divided by. (including any exp in 'd' type numbers)
420:
change number to correct sign if necessary.
421:
division to be done
no
(have answer already)
434
no
(To multiply by power of 10)
yes
JS 74 to divide by power of 10
419
JS 75 to multiply by power of 10
(with answer to 'read')
424
415
caption out SYMBOL 'char' INDATA
stop


417:
Erase '.'
'.' already encountered ie. end of number
yes
420
no
set decimal point counter to 0
(for next character)
412
418
418:
Erase 'α'
427
pick up first character of expression
ignore spaces between 'α' and exponent
char < 16
yes
(spurious)
415
no
char > 95
yes
(+,-, spurious)
428
no
429


429
429
accumulate expression
430:
pick up next character
char < 16 i.e. end of exponent
yes
no
yes
16 ≤ char ≤ 25
no
422
end of exp. i.e. number
change sign if necessary.
431:
decimal point counter positive
yes
423
(i.e. exponent is power of 10 to be used since no decimal point)
form total number to be used in multiplication or division sequence
423


428
425:
sign of exponent is positive
yes
no
sign of exponent is negative
no (spurious)
yes
set negative marker
435:
erase and go to pick up rest of exponent.
char < 16
yes (spurious)
no
16 ≤ char ≤ 25
yes
429
no
415


printfl (x, n)
fetch x
set up and standardise acc, x = f * 10^e, 1 ≤ f ≤ 10
round by adding ½ * 10^-n to f.
(result may be >10
subtract 10
f still >10
no
yes
divide f by 10 such that 1 ≤ f ≤ 10
increase e by 1
3:
print sign of x.
print 1 digit
fetch n
≤ 0
yes
(no frac pt)
no
print decimal point and n digits of f.
4.


434
434.
erase character
434A
floating number now in N1. fetch type of location 1 real, 2 integer
real
yes
(integer location)
no
floating point exponent negative i.e. number has frac. pt or is -1 or 0
yes
no
fix real quantity
number > 2^47
yes
(overflow)
8P:
437:
erase second copy of floating point exponent
frac pt ≠ 0
yes
no
(end sequence)
441
erase
436
move reconstructed line pointer back one and dump
Put 0 in DO of word
-1 or 0
yes
442
caption and real number found INSTEAD OF INTEGER IN DATA
no
STOP
dump number in location real to
fix -1 or 0
RETURN

```
P
DB72LNAACOMP#I
LOAD NEW AA COMPILER α
ST;
TL 3600;
V13;
PROGRAM;

V0=B0000000000160000;
V1=B0000000000177777;
V2=B0207414146415154; (AA FAIL);
V3=B0207624541447100;
V4=B6457006245414400;
V5=B6063000000000000; (READY TO READ PS);
V6=Q0/AV3/AV5;
V8=B0207436441604534; (CTAPE);
V9=Q0/AV8/AV9;
V10=B0207676441604534; (WTAPE);
V11=Q0/AV10/AV11;
V12=B0207544662575534; (LFROM);
V13=Q0/AV12/AV13;



V13; = Q1; TWQ1; (READ OLD COMPILER WORK TAPE);
V13; SET 4; OUT ; =C1; SET 5; =M1;
MRWDQ1; MFSKQ1; SET 14510; =I1; SET 14612; =M1;
SET 150; DUP; =RM5; =RM6; Q0 TO Q11;
SET 14511; =RM7; SET 14513; =RM8; ZERO;
(END OF INITIALISATION);

(COMPUTE BLOCK CHSM AND GO TO SWITCH);
1; MFRQ1; SET 100; =RC9; ZERO; VR;
3; M8M9Q; +; J2NV; NOT; NEG;
2; J3C9NZ; MOM7N; J199≠; +; J4NV;
NOT; NEG;
4; SET 100; =RC9; MOM7;
(SWITCH); NEG; NOT; DUP; J100<Z; DUP;J110=Z;
SET 2; -; DUP; J120<Z; DUP; J130=Z;
NEG; NOT; J140=Z; J170;
```

```
100; (PROGRAM); ERASE;
101; SET 100; M5; +; M6; -;
J104≤Z; JS10; J101;
*104; M8M9Q; MOM5; OR; =MOM5Q;
J104 C9 NZS; J1;
110; (LABELS); ERASE;
114; JS30;
111; DC3; DUP; J112=Z; JS40;
DUP; DUP; SET B7777; AND; CAB;
SET B10000; AND; SHL 7; CAB; VO;
AND; SHL 3; OR; OR; JS50;
113; J111 C3NZ; J114 C9NZ; J1;
112; ERASE; ERASE; J113;
120; (SETS); ERASE;
124; JS30;
121; DC3; DUP; J122=Z; JS40;
V1; AND; JS50;
123; J121 C3NZ; J124 C9NZ; J1;
122; ERASE; ERASE; J123;
130; (STACK); ERASE;
I11; J60≠Z; M5; =I11; J60;
140; (ROUTINE ADDRESSES);
M11; J60≠Z; I11; =M5;
MOM5; =+M5; M5 TO Q11; J60;
170; (PROGRAM CHECKSUM);
MOM8; VR; MOM7N; +; J171NV; NOT; NEG;
171; -; J199≠Z;
(ENTRY SEQUENCE);
M11 TO Q12; I11; =M5; MOM5N;
=+M12; I12=+1;
C1; SET 6; OUT;
V9; =Q1; TWQ1; (READ COMPILER TAPE);
V11; =Q1; TWQ1; (READ WORK TAPE);
V9; SET 4; OUT; =E418;
V11; V9; -; J172=Z;V11;SET 4; OUT ; DUP; =E419; =C15;
IO TO Q15; SET 1; =M15;
MRWDQ15; MFSKQ15;
MWQ15; MWQ15; MWQ15; MWQ15; J173;
172; E418; =E419;
173; SET 10; SET 5; OUT; =E420;
```

```
SET 9; SET 5; OUT; =E421;
ZERO ; NOT ; = E423 ;
VR; ZERO; =TR;
V6; =Q15; TWQ15; JE150;
(Q11 IS 0/AD ST(0)/AD RA(0));
(Q12 IS 0/1/WS);
(MESSAGE ON-LINE--READY TO READ PS);
199;(CHECKSUM FAILURE);
V2; =E0; TWQ0; ZERO; OUT;
(MESSAGE ON-LINE--AAFAIL--AND STOP);

10; (CLEAR SPACE);
SET 100; =C6;
*11; ZERO ; =MOM6Q; *J11 C6NZS; EXIT 1;
30; (UNPACK 2 WORDS);
M8M9Q; =Q2; M8M9Q; =Q3;
M3; I3; C3; M2; I2; C2; SET 3; =C3; EXIT 1;
40; (Q10 =SYL/-/ADD);
DUP; SET B17777; AND ; =M10;
SHL-13; SET 7; AND; =C10; EXIT 1;
50; (LOAD PATTERN TO ADDR IN Q10);
SHL 24; ZERO; REV; J51 C10Z;
*52; SHLD-8; DC 10; *J52 C10 NZS;
51; M6; M10; NOT; +; J53>Z; JS10; J51;
53; MOM10; OR; =MOM10; MOM10N; OR; =MOM10N;
EXIT 1;
*60; (DUMP STACK OR RA);
M8M9Q; =MOM5Q; *J60 C9NZS; J1;
FINISH; β
```

```
P
DB72CPR/AA/I
START OF AA RUN α
ST;
TL 60 ;
V40;
RESTART ; JE 425 ; JE 424 ;
PROGRAM;


V0 = Q0/0/0;
V1 = B0763644162640057;
V2 = B4600414100626556; (START OF AA RUN);
V3 = Q0/AV0/AV2;

V4 = Q0/0/0;
V5 = B0743575560515445;
V6 = B6200644160450034; (COMPILER TAPE TO V7);
V7 = Q0/0/0;
V8 = Q0/0/0;
V9 = Q-1/AV4/AV8;

V10 = Q0/0/0;
V11 = B0743606200675762;
V12 = B5300644160450034 ; (CPR WORK TAPE - TO V13);
V13 = Q0/0/0;
V14 = Q0/0/0;
V15 = Q-1/AV10/AV14;

V16 = Q0/0/0;
V17 = B0762454144456234; (READER);
V18 = Q0/0/0;
V19 = Q-1/AV16/AV18;

V20 = Q0/0/0;
V21 = B0751566065640055;
V22 = B4147006441604534; (INPUT MAG TAPE);
V23 = Q0/0/0;
V24 = Q0/0/0;
V25 = Q-1/AV20/AV24;
```

```
V26 = Q0/0/0;
V27 = B0707606556435034; (PUNCH);
V28 = Q0/0/0;
V29 = Q-1/AV26/AV28;

V30 = Q0/0/0;
V31 = B0707075451564500;
V32 = B6062515664456234; (LINE PRINTER);
V33 = Q0/0/0;
V34 = Q-1/AV30/AV33;

V35 = Q0/0/0;
V36 = B0757656460640055;
V37 = B4147006441604534; (OUTPUT MAG TAPE);
V38 = Q0/0/0;
V39 = Q0/0/0;
V40 = Q-1/AV35/AV39;

V3; SET 8; OUT;
V9; SET 8; OUT;
V15; SET 8; OUT;

V7; V13; J1 = ;
V13; SET 4; OUT; DUP; = C1; DUP; =C2; =E419;
SET 1; = M2; MRWDQ2; MFSKQ2;
ZERO; DUP; = I1; = M1;
MWQ1; MWQ1; MWQ1; MWQ1;
SET 4; OUT ; J2;
1; SET 4; OUT ; DUP; = E419;
2; DUP; =E418; =C15;
SET 2; = M15; MRWDQ15; MFSKQ15;
SET 8; =I15; SET 100; = M15;
SET AR3; =M14; MOM14; =E150;

6; V19 ; SET 8; OUT ;
V18 ; ZERO ; SHLD+6 ;
SETB 71 ; (Y) ; J4 = ;
SETB 56 ; (N) ; - ; J5 = Z;
ERASE ; ZERO ;
=V16 ; J6 ; (NEITHER Y NOR N) ;
5; ERASE ; V25 ; SET 8 ; OUT ;
V23 ; SET 4 ; OUT ;
DUP ; = C1 ; SET 1 ; = M1 ;
MRWDQ1 ; MFSKQ1 ; J7 ; (MAG INPUT);
```

```
4; ERASE ; ERASE ; SET 10 ; SET 5 ; OUT ;
7 ; = E420;

10; V29 ; SET 8 ; OUT ;
V28 ; ZERO ; SHLD+6 ;
SETB 71 ; (Y) ; J8 = ;
SETB 56 ; (N) ; - ; J9 = Z ;
ERASE ; ZERO ;
= V26 ; J10 ; (NEITHER Y NOR N) ;
9 ; ERASE ; ZERO ; J11 ;
8 ; ERASE ; ERASE ; SET 9 ; SET 5 ; OUT ;
= E421 ; ZERO ; NOT ;

11 ; V34 ; SET 8 ; OUT ;
V33 ; ZERO ; SHLD+6 ;
SETB 71 ; (Y) ; J12 = ;
SETB 56 ; (N) ; - ; J13 = Z ;
ERASE ; ZERO ;
= V30 ; J11 ; (NEITHER Y NOR N) ;
13 ; ERASE ; DUP ; J14 = Z ;
= E423 ; (PUNCH ONLY) ; J15 ;
12 ; ERASE ; ERASE ; SET 11 ; SET 5 ; OUT ;
= E422 ; NOT ; NEG ; = E423 ; (BOTH OR LP ONLY ) ;
J15 ;

14 ; V40 ; SET 8 ; OUT ;
V38 ; SET 4; OUT ;
DUP ; = C1 ; SET 1 ; = M1 ;
MRWDQ1 ; MFSKQ1 ; = E421 ;
NOT ; =E423 ; (MAG OUTPUT) ;

15 ; ZERO; =LINK; JE150;
*3; MFRQ15; EXIT 61;

FINISH; β
```

```
P
DB72PTFCAA/I
PREPARE TAPE FOR VERSION I α
ST;
TL 60;
V21;
PROGRAM;


V0 = Q0/0/0;
V1 = B0707076064464334; (PTFC);
V2 = Q0/0/0;
V3 = Q-1/AV0/AV3;
V4 = Q0/0/0;
V5 = B0707414100604162 ; (AA PAR);
V6 = Q0/AV4/AV5;

V7 = Q0/0/0;
V8 = B0707414100636555; (AA SUM);
V9 = Q0/AV7/AV8;

V10 = Q0/0/0;
V11 = B0760625747625500; (PROGRM);
V12 = Q0/0/0; (FOR TIME);
V13 = Q0/AV10/AV12;

V14 = B1212010612010612; (RADIX WORD);
V15 = B2020372020372020;

JE 150 ;
(ON ENTRY TAPE IS 3 BLOCKS IN, DEVICE NO IN C15) ;
(DEVICE NUMBERS IN E418-23) ;
11; J10EN ; ERASE ; J11 ; 10 ; J12EJ ; LINK ; ERASE ; J10 ;
12 ; E418 ; = V16 ; E419 ; = V17 ; E420 ; = V18 ; E421; = V19 ;
E422 ; = V20 ; E423 ; = V21 ;

V14; SET 9 ; OUT ; SHL-24 ;
FRB ; V15 ; OR ;
= V12 ;
V13 ; SET 8 ; OUT ;
```

```
C15 TO Q14 ; SET 1 ; = M15 ; SET 101 ; = I14 ;
E1 ; SHL24 ; SHL - 24 ; NEG ; NOT ; = M14 ;
MFSKQ15 ; MFRQ14 ; PARQ14 ; J1TR ; MRWD Q15 ;
E146 ; =Q14 ; M14 ; SET 101 ; DUP ; = M13 ; - ; = RC14 ; ZERO ; ZERO ;
*2 ; M13M14Q ; * STR ; +D ; J2 C14 NZS ;
M13M14N ; M13M14 ; -D ; J3 ≠ Z ; J3 ≠ Z ;

V16 ; = E418 ; V17 ; = E419 ; V18 ; = E420 ; V19 ; = E421 ;
V20 ; = E422 ; V21 ; = E423 ;

ZERO ; =E434 ;

SET 118 ; = M13 ; MOM13 ; DUP ;
= C13 ; = + M13 ; I13 = -1 ; J7C13Z ;
6 ; MOM13Q ; = LINK ; J6C13NZ ;
7 ; SET 135 ; = M14 ; SET 15 ; = RC13 ;
*8 ; M14M13Q ; * J8C13NZS ;
= Q15 ; = Q14 ; = Q13 ; = Q12 ; = Q11 ;
= Q10 ; = Q9 ;  = Q8 ;  = Q7 ;  = Q6 ;
= Q5 ;  = Q4 ;  = Q3 ; = Q2 ;  = Q1 ;
SET 101 ; = M13 ; MOM13 ; DUP ;
= C13 ; = + M13 ; I13 = -1 ; J4C13Z ;
5 ; MOM13Q ; J5C13NZ ;


4 ; V12 ; = E427 ;
VR ; ZERO ; = TR ; EXIT 1 ; (TO COMPILER) ;

1 ; V6 ; J9 ; 3 ; V9 ;
9 ; SET 8 ; OUT ; ZERO ; OUT ;

RE150 ; V3 ; SET 8 ; OUT ; V2 ; SET 4 ; OUT ; DUP ; = C15 ;
DUP ; = C14 ; = C13 ;
SET 1 ; = M15 ; MRWDQ15 ; MFSKQ15 ; (SKIP LABEL) ;
SET 7 ; DUP ; = I15 ; = M15 ;
SET 8 ; = I14 ; SET 100 ; = M14 ;
SET 115 ; = M13 ;
MWQ15 ; MWQ14 ; MWIPEQ13 ; MWQ15 ; MWQ15 ;
C15 ; SET 6 ; OUT ; ZERO ; OUT ;
(WRITE SENT, CALL, SENT, DUMMY) ;
(DIRECTOR CHECKS PARITY ON REWIND) ;

FINISH ; β
```

To Mr.H. Whitfield, Computer Unit, 7 Buccleuch Place, 8.

KDF 9 MAGNETIC TAPE ALLOCATION REQUEST

Please allocate to me _____ sections of 512 words each on a magnetic tape at Glasgow University Computing Laboratory.

Date __/__/__ Signed _____

Dept. _____

LAYOUT OF MAGNETIC TAPE FOT USE WITH MAG. TAPE ROUTINES

THIS DOES NOT APPLY TO COMPILER TAPE OR WORK TAPE

0		1		2	
L	S	D	S	B1	S	B2

L	LABEL BLOCK	
S	SENTINEL BLOCK	ONE WORD
D	DICTIONARY	512 WORDS
B	USERS' BLOCKS	512 WORDS

D(0) - D(511) are dictionary words.

Di marker/first block/last block (i is reduced parameter)

marker = 0 code not allocated on this tape

$\neq 0$ allocated

PARAMETER * d1 d2 d3 d4 d5 d6 d7 d8

reduced parameter = $8^2 \times d1 + 8 \times d3 + d5$

d7 = (d1 + d3 + d5)modulo 8

d2 = d1 $\not\subset$ 1 d4 = d3 $\not\subset$ 2 d6 = d5 $\not\subset$ 3 d8 = d7 $\not\subset$ 4

MAGNETIC TAPE CONTROL COMPILERS F and G

89P:

	REDUCED PARAMETER	
ch. 1	IDENTIFIER	TAPE WORD
2		
3		
4		
5		
6		
7		
8		

INITIAL VALUES:-

PARAMETER 0
IDENTIFIERS -1
TAPE WORDS

TAPE WORD

± device	first block	last block

device no. > 0 read only
< 0 write permit

ON A SPECIFIED CHANNEL IDENTIFIER IS SET AND TAPE WORD IS ± 1 TO INDICATE MODE.

WHEN CHANNEL IS CLAIMED TAPE WORD IS FILLED AS ABOVE.

DATA LAYOUT FOR ALLOCATION OF TAPE SECTIONS PROGRAM VER. G

Example:-

50*Fred*DG720004*01142136

No spaces:

```
P
DB720001MGAD
ADDRESS MAG TAPE α
ST;
TL 3600;
V5;
PROGRAM;


V0=B0207006441604534;
V1=Q0/AV0/AV1;    (TAPE);
V2=B1212121212121212; (RADX WD);
V3=B0207544163644254; (LAST BL);
V4=B0000002020202020;
V5=Q0/AV3/AV4;


V1; = Q1; TWQ1; (IDEN TO V1);
V1; SET 4; OUT;
DUP; DUP; = C13; = C14; = C15;
SET 1; = M13; MRWDQ13; MFSKQ13;
(PICK UP TAPE AND SKIP LABEL);

SET 513; = RC1; ZERO;
* 1; DUP; = YOM1Q; * J1C1 NZS;
\\\\SET AY\\\\\\\0; = I13; SET AY511; = M13;
SET AY512; DUP; = I14; = M14;
SET 255; = M15;



3; MWQ14; MWQ13; MWIPEQ15;
METQ13; J2TR;
NOT; NEG; DUP; = Y512; J3;
(COUNT IN N1 AND Y512);


2; V2; REV; FRB; V4; OR; = V4;
V5; = Q1; TWQ1; (LAST BL OO\\186);
C13; SET 6; OUT; ZERO; OUT;
(DIRECT\\OR CECKS PARITY REWINDING);
FINISH; β
```

```
***A
JOB
COMPILER GROUP HW ALLOCATE TAPE SECTIONS
COMPILER AA


mcode


\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\


begin
integer i,j,first,last,code,sections
integer array D(0:511)

1:read(sections)
caption ħ start $ new $ page $each $ time $ tape $ stops ħ
stopcode
newline
spaces(30)
caption Computer $ Unit,
newline
spaces(33)
caption University $ of $ Edinburgh,
newline
spaces(36)
caption 7$Buccleuch$Place,8.ħħ Dear$
readsymbol(i)
stop unless i='*'
2:readsymbol(i)
-> 3 if i='*'
printsymbol(i)
->2
3:caption ,ħ$$$$You$have$been$allocated
print(sections,1,0)
caption $ sections  $numbered$1$to$
print(sections,1,0)
caption$on$aħmagnetic$tape$held$ at$Glasgow$University$ Computing$Laboratory.
caption ħħTAPE$IDENTIFIER$$
*ZERO
cycle j=1,1,8
*SHL+6
readsymbol(i)
printsymbol(i)
**i
*SET63
*AND
*OR
repeat ; |identifier is in MS
```

```
stop unless nextsymbol='*'
caption ♮PARAMETER$$$$$$$*
skipsymbol
code=0
cycle j=1,1,4
readsymbol(i)
printsymbol(i)
code=8*code +i-16
**i
readsymbol(i)
printsymbol(i)
**i
*NEV
**j
*-
*J 13≠Z
repeat
**code
*SHL-3
**=code
newlines(2)
spaces(20)
captionSigned.....
caption ♮♮Entered$on$tape$file$on$
*SET7
*=M13
*MOM13
cyclei=1,1,8
*ZERO
*SHLD+6
**=j
printsymbol (j)
repeat
*ERASE
newlines(3)
stopcode
```

```
*       SET 4
*       OUT
*       DUP
*       = C15
*       = C14
*       SET 2
*       = M15
*       * αD(0)
*       = I14
*       * αD(511)
*       = M14

*       PMD Q15         ; | rewind
*       PMA Q15         ; | skip label and sentinel
*       PIA Q14         ; | read dictionary
        -> 4 unless D(code) = 0 ; | code already assigned on this tape

*       * D(0)
*       = Q13
*       I13
*       DUP
*       NOT
*       NEG
*       = I13   ; | first section
*       * sections
*       +
*       DUP
*       = M13   ; | last section
*       ZERO
*       NOT
*       NEG
*       = C13
*       Q13
*       * = D (code)
*       * D(0)
*       = Q13
*       = I13
*       Q13
*       * = D(0)
*       SET 65
*       = M13
*       C15 TO Q13
```

```
*       PMD Q15         ; | rewind
*       PMA Q15         ; | skip label and sentinel
*       POA Q14         ; | write dictionary
*
*       54334/7417/7417         ; | write gap
*       PMD Q15         ; | rewind
*       PMA Q15         ; | skip label and sentinel
*       PIA Q14         ; | read dictionary
*       C14 ; * SET 6 ; * OUT
caption ↑\\\↑ DICTIONARY ↑↑ CODE ↓↓ FIRST ↓↓ LAST
        cycle i = 1,1,511
*       * D(i)
*       = Q15
*       J5C15Z
*       I15
*       *= first
*       M15
*       * = last
        newline
        print (i,3,0)
        print(first,6,0)
        print(last,5,0)
5:      repeat

        caption ↑↑ LAST ↓ SECTION ↓ ALLOCATED
*       * D(0)
*       = Q15
*       I15
*       * = last
        print(last,5,0)
        -> 1
4:*     C14
*       SET 6
*       OUT
        caption ↑ CODE ↓ ALREADY ↓ ALLOCATED
        print (code,1,0)
13:stop
end of program
```

PRODUCE CODE TABLE.

```
begin
newlines(4)
caption ßßßßß SECURITY ß CODE ß TABLE ßßßß No ßßßßß NAME ßßßßßßßßßßßßßßßß CODE ßßß
accept m/c instructions
integer array A(1:8)
integer i,j,k,m
cycle i=5,5,250
newline
** i
cycle j=3,-1,1
* DUP
* SET +7
* AND
* DUP
** =A(2j-1)
** j
* NEV
** =A(2j)
* SHL -3.
repeat

* ERASE
** A(1)
** A(3)
* +
** A(5)
* +
* SET +7
* AND
* DUP
** =A(7)
* SET 4
* NEV
** =A(8)

print(i,3,0)
caption ßßß .............ßß
cycle j=1,1,8
print(A(j),1,0)
repeat
repeat
end of program
```