PART OF GOTO ???

find certain line no.

```
C62E  DEC #04          ;} prepare to pull item from stack
      LDX #04          ;
      LDY @0           ;
      STY #58          ;} set up (#58) to start of text space
      LDA #12          ;
      STA #59          ;
      DEY              ; prepare to enter loop
C63B  LDA @#0D         ; and prepare to find a [CR]
C63D  INY              ;} set up (#58),Y to first [CR]
      CMP (#58),Y      ;
      BNE #C63D        ;
      JSR #CEA1        ;add Y reg. into pntr (#58), and inc pntr (test ESC key)
      LDA (#58),Y      ;} get byte (line number)
      INY              ;
      CMP #25,X        ;and compare with 'high' byte on stack
      BCC #C63B        ; if 'line found' < 'line to be found', then 'next line'
      BNE #C660        ; if " " > " " " " , return w/v 'carry set'
C64E  LDA (#58),Y      ;get other byte of line no.
      CMP #16,X        ;compare w/v 'low' byte on stack
      BCC #C63B        ;} AS ABOVE
      BNE #C660        ;
      STA #01          ;} hey kiddies, we've found line no., so put line
      LDA #25,X        ;      no. in #01,#02
      STA #02          ;
      JSR #CEA1        ;add Y reg to (#58), and inc (#58),Y
      CLC              ; clear carry to indicate 'LINE FOUND'
C660  RTS              ; and FINISH
-------------------------------------------------------------------
C661  JSR #C88C        ;evaluate <expression>
      LDA #42,X        ;} EOR signs on the 2 TOS items
      EOR #41,X        ;
      STA #52          ; save result (= sign of result of * or /
      JSR #C905        ;if TOS is -ve, then negate it
      LDY @#53         ;} pull item from TOS and put in !#53
      JSR #C3CD        ;
      LDA #42,X        ; if TOS is -
      STA #43,X        ;
      JSR #C907        ;if TOS is -ve, then negate it
      LDY @#57         ;} pull item from TOS and put in !#57
      JSR #C3CD        ;
C67E  LDY @0           ;
      STY #5B          ;
      STY #5C          ;} zero 32 bit accumulator !#5B
      STY #5D          ;      which will later hold result
      STY #5E          ;
      RTS              ;and FINISH
-------------------------------------------------------------------
C689  JSR #C661        ;evaluate expression, RESULT←0, dividend→!#57, divisor→!#53
      LDA #54          ;} is divisor = 0?
      JSR #C705        ;
      BEQ #C67F        ;if so, then ERROR
      LDY @#20         ;count = 32 (we're dealing w/v 32 bit numbers, FIDDLES!)
C695  DEY              ;]
```

C67F → BRK division by zero

```
C696    BEQ #C6D9      ;) do count = count - 1, until done 32 times
                                            or msbit
        ASL #57        ;}
        ROL #58        ;} shift left dividend } keep shifting dividend
        ROL #59        ;}                       until msbit = 1
        ROL #5A        ;}
        BPL #C695      ; if msbit = 0, branch
C6A2    ROL #57        ;}
        ROL #58        ;}
        ROL #59        ;}
        ROL #5A        ;} else shift dividend into RESULT
        ROL #5B        ;}
        ROL #5C        ;}
        ROL #5D        ;}
        ROL #5E        ;}
        SEC            ;}
        LDA #5B        ;}
        SBC #53        ;}
        PHA            ;}
        LDA #5C        ;} pretend to do (RESULT:= RESULT - DIVISOR)
        SBC #54        ;}
        PHA            ;}
        LDA #5D        ;}
        SBC #55        ;}
        TAX            ;}
        LDA #5E        ;}
        SBC #56        ;}
        BCC #C6D4      ; branch if borrow (i.e. RESULT < DIVISOR)
        STA #5E        ;}
        STX #5D        ;}
        PLA            ;} do RESULT := RESULT - DIVISOR
        STA #5C        ;}
        PLA            ;}
        STA #5B        ;}
        BCS #C6D6      ; and repeat loop (REM : CARRY SET!)
C6D4    PLA            ;} remove 'false data' from machine stack
        PLA            ;}
C6D6    DEY            ; count := count -1
        BNE #C6A2      ; do for all bits
C6D9    RTS            ; And return
-----------------------------------------------------------------------
C6DA    JSR #C78B      ; evaluate unbracketed expression, X <- stack ptr
        DEX            ;} prepare to pull item from stack
        STX #04        ;}
        LDA #42,X      ;}
        EOR @#80       ;} #52 <-
        STA #52        ;}
        LDA #43,X
        EOR @#80
        STA #54
        LDY @0
        SEC
        LDA #15 X
```

? = undefined
1 = set
0 = cleared
S = same as entry

ROM ROUTINES

F291 :- skip spaces and get char [A,Y,F]

ENTRY (#05)+?#03 points to character with which to start with

EXIT Acc ← first non-space character
(#05), Y points to position of char.
(#05)+?#03 points to immediately AFTER char.

N	V		B	D	I	Z	C
?	S	-	S	S	S	0	S

F36B :- sets up pointer (#52) to variable P [A, F]

ENTRY None

EXIT (#52) ← P
Acc. = high byte of (#52)
if (#52) dont point to zero page, Z clear

N	V		B	D	I	Z	C
?	S	-	S	S	S	?	S

F376 :- print hex contents off acc. followed by a space [A, F]

ENTRY acc ← number
D flag ← 0

EXIT acc ← #20

N	V		B	D	I	Z	C
0		-	S	S	S	0	?

F37E :- print hex contents off acc. [A, F]

ENTRY acc ← number
D clear

EXIT acc ← ASC II char. of lowest number i.e. if acc = #35, acc ← CH "5"

N.B. unlike routine #F802, this routine keeps 'count' correct!

F38E Assemble mnemonic