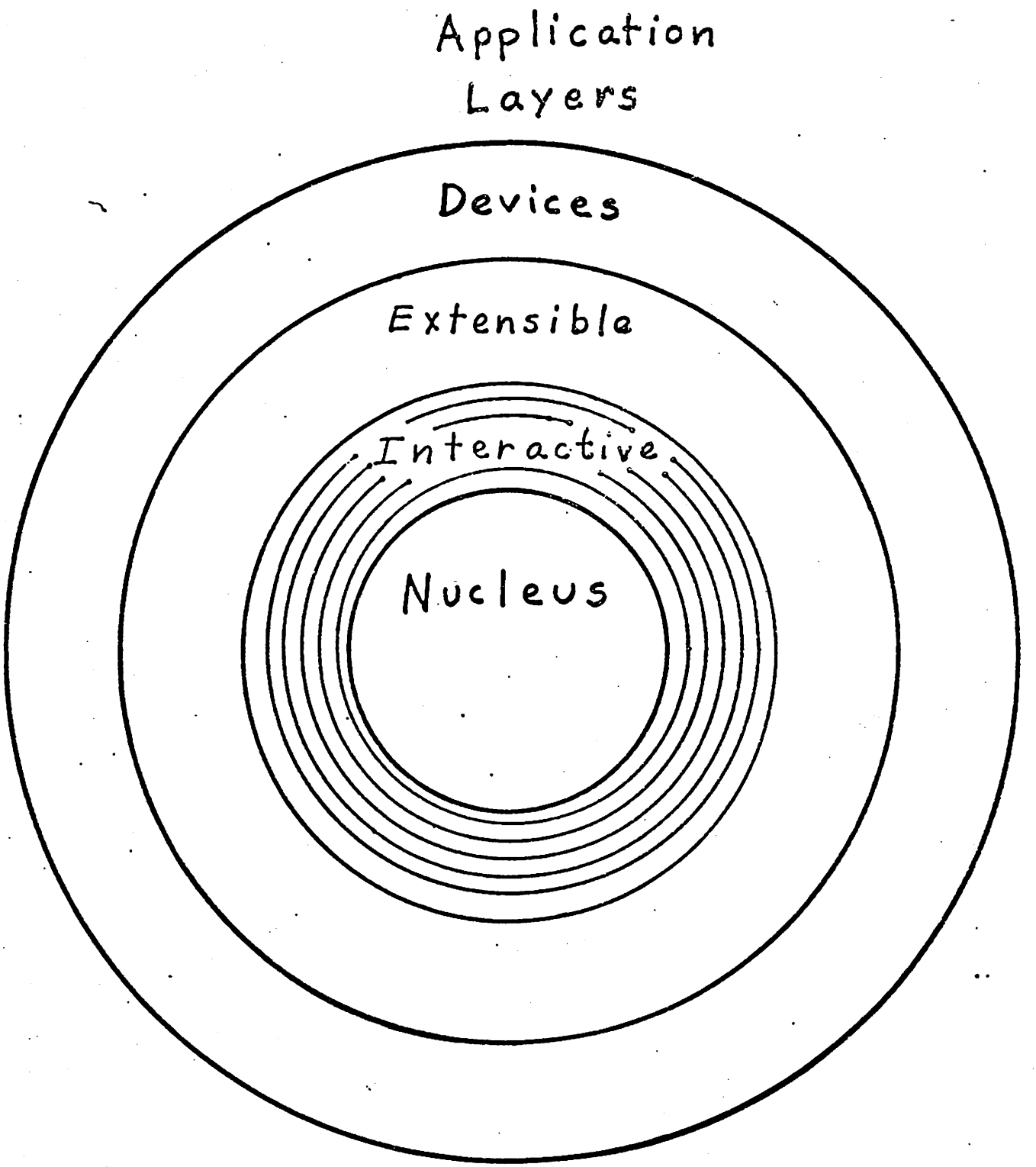
STRING OPERATIONS



STRING HANDLING

Input

Conversion

- Numeric string ↔ binary
- Copying
- Formatting

Comparison

Output

STRING INPUT

Read a string of characters from your terminal (or a communications channel)

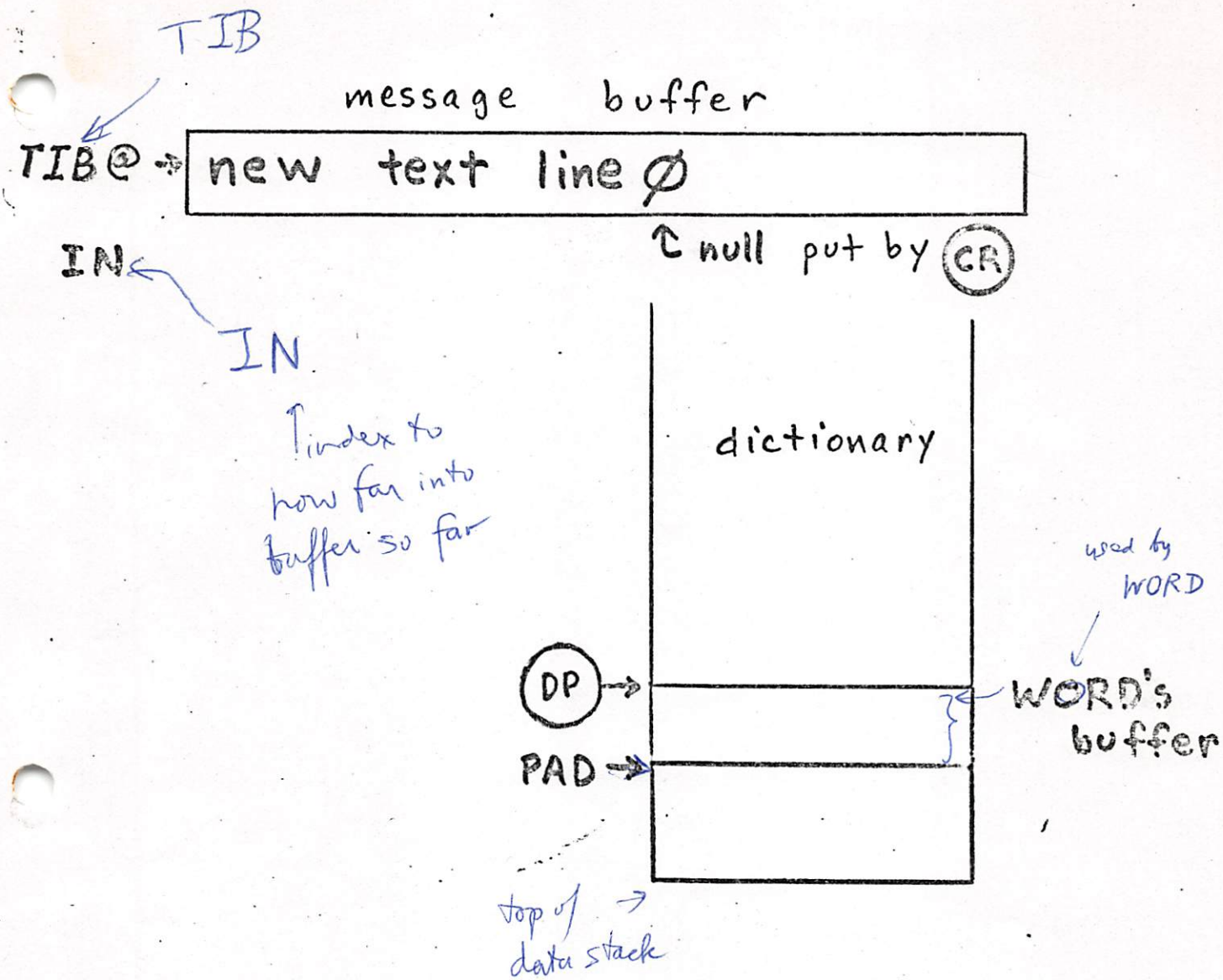
dest_addr max#chars EXPECT

in FIG, built as loop on KEY

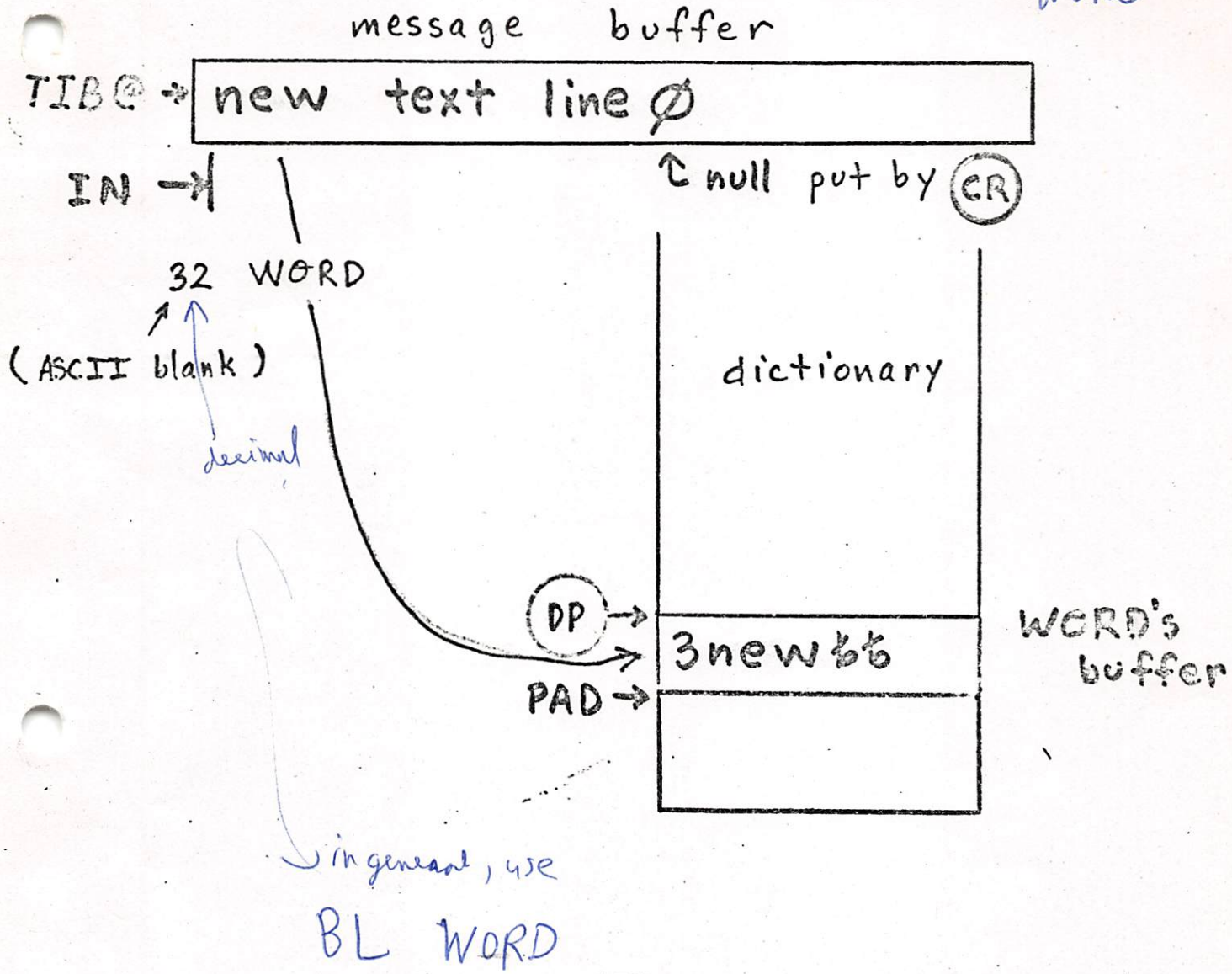
in FORTH Inc, based on interrupts & KEY is 1 EXPECT

performs BackSpace editing

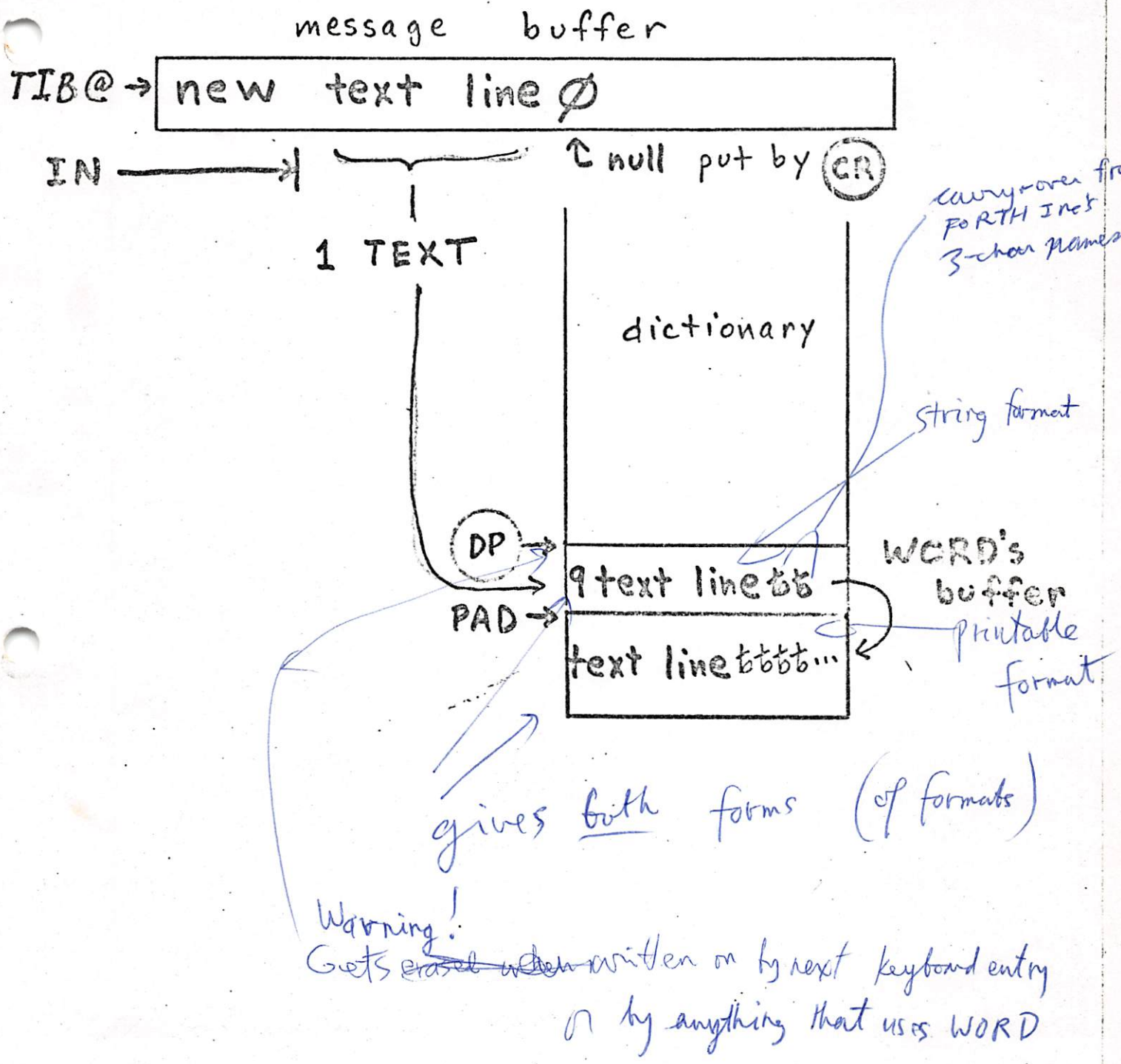
terminates when { max# chars entered
or
(CR) entered


TIB
message buffer
TIB@
new text line ∅
null put by CR
IN
IN
index to how far into buffer so far
dictionary
used by WORD
DP
WORD's buffer
PAD
top of data stack

Action of WORD

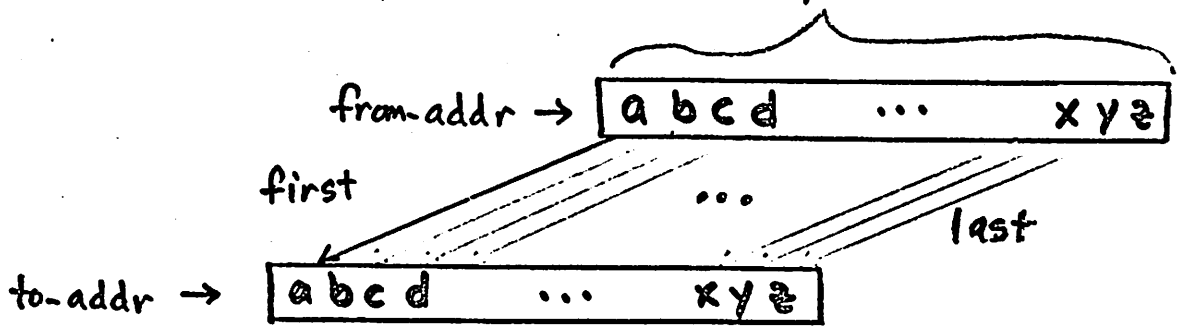


Action of TEXT



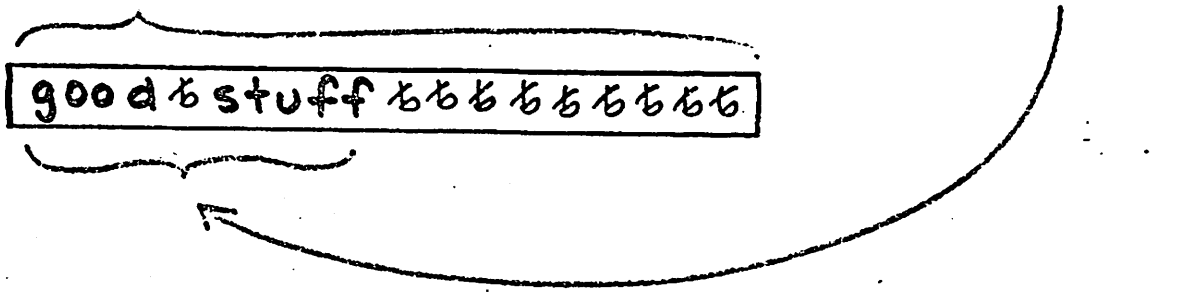
Copying strings

from-addr to-addr #bytes MOVE



Remove trailing blanks

addr before#bytes -TRAILING --- addr after#bytes



Initialize strings (or arrays)

addr #bytes BLANKS stores blanks

addr #bytes ERASE stores zeros

addr #bytes character FILL stores "character" starting at "addr" for "#bytes"

STRING OUTPUT

SS
rd
fig

Write characters to
your terminal
(or a communications line)

FORTH INC. : interrupt driven routine

addr #bytes **TYPE** write string — FIG : TYPE is loop on EMIT

FORTH INC: 1 TYPE

chr_value **EMIT** write single character

CR write Carriage Return (Line Feed)

SPACE write single blank

#spaces **SPACES** write several blanks

." wow " write string

Example

```
: ECHO  TIB @ 80 EXPECT
        0 IN !
        BEGIN (optional)
        TEXT letter WORD
        TYPE CR 0 END
        HERE COUNT
```

can call QUERY

can also use

```
QUIT
QUERY
INTERPRET
." OK" CR
```

a "forever loop"

STRING CONVERSION

ASCII character to numeric value:

use:

ASCII chr --- { chr_value if interpreting
else compiling, then
chr_value is compiled
as a 16 bit literal

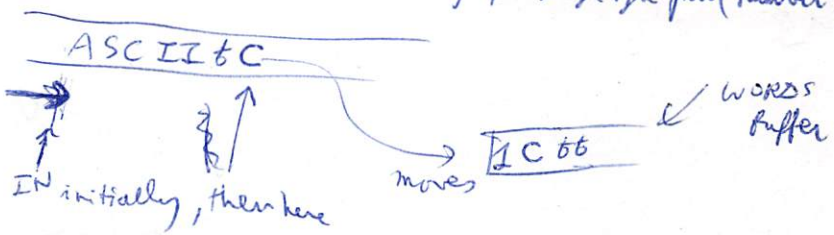
definition:

```
: ASCII    BL WORD    HERE 1+ C@
           LITERAL ;  IMMEDIATE
```

? makes wait till we learn about compilers

← Compiles into dictionary the ~~single~~ byte pair (number)

On execution,



Numeric string to binary conversion:

addr_string **NUMBER** --- dvalue

and following conversion,
variable **DPL** contains

-1 if numeric string was not punctuated (converted value is in the 16 bit signed integer range)
else the number of digit characters following the last punctuation
(32 bit converted value)

see p. 60 USING FORTH (old version) for 4 ways to
print 32 bit, 31 bit
16 15 bit
no's

use of HOLD to stick in special char's in format
e.g. print 400.0 deg °C in our temperature conversion example

```
: .DEG  SWAP OVER DABS
   <# # ASCII . HOLD #S  SIGN #> TYPE SPACE
```

(loop around #

Note: can put any other "action" here

e.g. $ XX, XXX, XX.XX without leading 0's

Building the word " . "

e.g.

ABS ⊙

will print a leading sign

DABS SIGN

```
: .DUP S->D <# #S #> TYPE ;
```

need copy of signed value

see p.51 USING FORTH

So if you want financial figures specially formatted

Pictured numeric output conversion:

Convert 123_{10} to a numeric string:

$$\frac{123}{10} = 12 \text{ remainder } \boxed{3}$$

$$\frac{12}{10} = 1 \text{ remainder } \boxed{2}$$

$$\frac{1}{10} = 0 \text{ remainder } \boxed{1}$$

Convert unsigned 32-bit value to 10 digits:

uses PAD

```
: UD.10   <# # # # # # # # # # # #>
  TYPE ;
```

leaves what TYPE wants

- `<#` starts numeric conversion
- `#` converts one digit
- `#>` ends conversion

only works on pos #'s

```
3.1415928  UD.10 CR  0031415928 OK
```

#S will do a complete conversion of the number string on the stack

eg. `<# #S #>`

```
: #S BEGIN # DUP 0. = UNTIL
```

16 bit version

see Scr. 75 for 32 bit version

USER allows "private" variables

e.g. offset USER BASE

- creates a single head in dictionary
& execution time procedure which adds the offset of the current user

MEMORY ALLOCATION