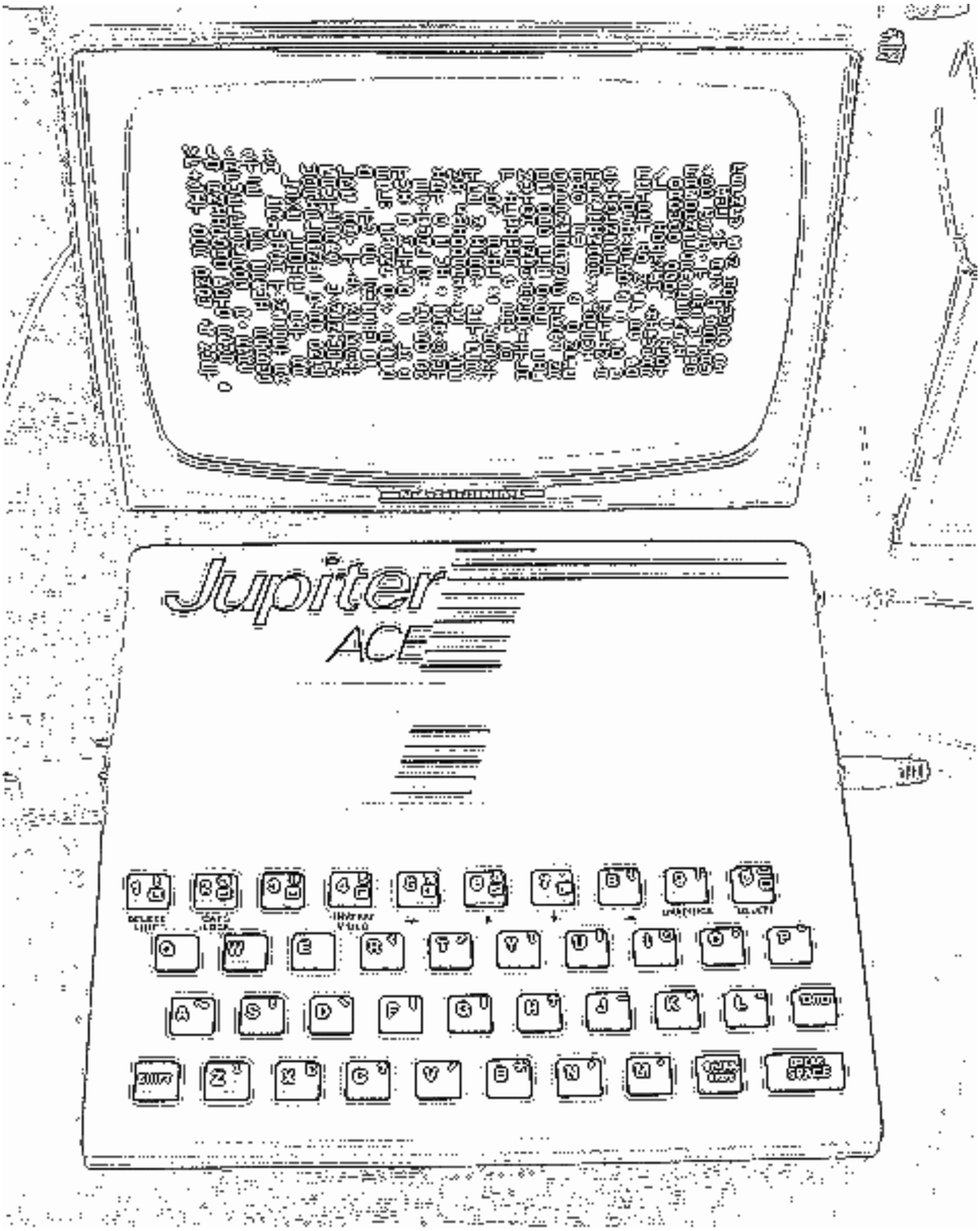
ACE ROM Documentation Proj *(2022.252)*



= An ACE back, and Forth =

(The Greatness of the Humble)

• •

Details:

Document Date : 2022/Jun @LuxBonna, Lusitanea by the Sea.
Building Tools: An ultra-slim, lousy keyboard, small screen, dangerous touchpad.
Language used : International English, irrelevant form, weaved as a patches blanket.
This author learned to write and read a precise latin derived language
with plenty of Japanese (Nagasaki) and Indian influences (India, not Texas).

To whom it may regard: (or decipher)
The above are suppositions. Never opinions nor theories. Not even when joking.
To whom enjoys "to file", no longer with the care placed down "the rabbit's hole":
As the 'shaker' would had said today: "To file or not to file. Ask not, roll the dice".
Each paragraph is a place, with its own rules and consistence. Coherence is the pudding.

Legal Status, as enforced wherever GPL is applied or adapted:

Availability:

While now residing of https://t.me/JupiterAce the very last ACE-ROM_Doc_Prj edition can be found at https://drive.google.com/file/d/1ykjRsfCsfSOKw1YcOH6SyDjZT8hwUxtn/view?usp=sharing

A simple wish:

We wished the information here was available 30 years ago.
Or maybe not. Each one's path is for each one! (It is)
Though it was a bumpy path. A path is here cleaned.

And a prelude quote:

When most I wink, then do mine eyes best see,
For all the day they view things unrespected;
But when I sleep, in dreams they look on thee,
And darkly bright, are bright in dark directed.

Then thou, whose shadow shadows doth make bright,
How would thy shadow's form form happy show
To the clear day with thy much clearer light,
When to unseeing eyes thy shade shines so!

How would, I say, mine eyes be blessed made
By looking on thee in the living day,
When in dead night thy fair imperfect shade
Through heavy sleep on sightless eyes doth stay!

All days are nights to see till I see thee,
And nights bright days when dreams do show thee me.

~ (Quite True 43 years back, before this recall)

A few End-Of-Page taglines, to temper page reading:
To teach, is to point. Pointers available, not solutions.
To own is to do (understand)... Never tag or attribution.
Tags are just tags... Good and bad, all do have a chance.

Dedicatory

This e-book is dedicated to:

All those Mahatmas unknown,
who lived regular lives, common or not.
Some dead by the hands of those they protected.

All the brave souls, known or not,
who kept their lives integral, not a fraction.
Then kept and raised the soul of a Mankind to come.

Both here, for a while. Teaching and protecting.
They remind what we all forgot. As their names were.
Killed by induced diseases, not known, nor the dirty why.

They are alive, quite high. Above the the hearts of many.
May all the humble then rise, not on Ego nor Pride.
Remembering why we all have born here, and now.

Let us find whom! On this empire of Maya.

A short description of this work:

ACE-Forth could had been one of the best ‘thing’ that ever happened to Forth.
It was available to common people on a cheap package. An opportunity lost.
Wonder who the secret investor would be, shooting himself in the foot.
(Has damaged such chances...In the protection of what interests?)

Life, Questions... Overlooked Results:

How can my Muse want subject to invent?
Be thou ten times more in worth
Than those he calls, let him bring Forth.
The pain be mine, but thine shall be the praise.
~ ShakeForth (in a parallel testimony)

ACE-Forth also was silently pre-OOP and OOP ready (efficiently) at a time
those notion were not known. We can only imagine the progresses denied
both to an entire country, and continent..Now replaced by ‘dependency’.
We have felt such unseen benefits, we have placed in practice.

Here we share restoring code, after a small restored ROM.
We give hints when more would be out of the scope of this sharing.
Knowing how the world works in the hide, we may already have given too much.
Maybe... Hope not (believe nothing).

Enjoy this sharing.

Look Back, Look Forth.
There are chances all around. Let them not be wasted *(Do not try, be those)*.
“Rather fail with honour than succeed by fraud”. ~ Sophocles

___
 # Forgive all, forget not... Believe nothing, look forth...
"The trick, William Potter, is not minding that it hurts" ~ T.E. Lawrence

Full Content

Discovering Forth...

... On a Computer-Nostalgia adventure

An ACE back, and Forth

(Adventures with a language that could)

General description

+ JUPITER Architecture
+ ACE FORTH Benchmarks

ACE-Forth details

+ Programming Tips
+ Building Tools
+ A Prims Libs

ROM Original Listing

+ Restored back
+ Z80 Codes&Clocks

FORTHs internals

+ Inner Concepts
+ Sequencing 'Modes'
+ Structural Subjects

by Dutra de Lacerda
"The Human Factor Workshop"
now on https://t.me/JupiterAce

Special Thanks

To the following *(with date of usage)... Highly recommended*
... Starting much curiosity on FORTH mechanics. Enhancing its alchemy, step-by-step.

1984 ! *Stephen Vickers* - the Perfectionist, for the ACE-FORTH tool (1982)
The culprit of all this, giving us a beacon... After *C.Moore* - the Dreamer.
2011 ! *R.G. Loeliger* - the Experimenter, for “Threaded Interpretive Languages” (1981)
An engineering encyclopedia of WHAT & HOW perspectives (freedom and clever code).

2013 *Dr. C.H.Ting* - the Examiner, for "Inside F83" 3th Ed (1991)
Exposing implementation details and including the clever Multi-Tasking.
2017 *Leo Brodie* - the Player, for "Thinking Forth" (1984). A too late read.
Showing FORTH essence (the many ways to build a canoe). A sure joy.

2010 *Brad Rodriguez* - the Copier, for its "Moving Forth" series (1993)
Reveals various dispatchers, registers choice, the DOES> rationale.
2015 *Anton Earl* - the Dancer, for articles on Structs/Records and resulting OOP
Dick Pountain - the Popper, for ”Object Oriented Forth" (1987)

Also to a few Publishing Magazines:
1979 MICRO SYSTEMES = A very informative French magazine, but expensive
1982 BYTE Magazine = Splendid trough the 80's, not surviving the 90's
2015 FORTH Dimensions Magazine = An historical repository: 1978-1999
2020 The Computing Journal = A fair BYTE-Mag alternative, 1983-1998

Then to all who search and try:
Whom are curious for more than 5 minutes, and who's silence is inquisitive.
In short: To those who ask themselves, no longer dressing postures to show.

Content State:
This document data is (mostly) correct at the time of writing. So, any failure (incompletion) may be fulfilled in the future.(iff considered adequate. i.e. of general interest and useful)

We apologise in advance (not much) for the inevitable editing glitches that may (hopefully) be corrected as fast as detected. We request readers to communicate their own findings.
... In fair retribution (i.e, sharing to assist the shared).

P.S. Thanks also, to Dr Strange Humour, for his invaluable advices. Not forgetting the readers who will bear spell glitches on a foreign language hardly written.

Preface to version 3

Why to write about a dead Computer? One few have experienced? Or to write code now anachronic?
Though not exactly for rookies, we share here plenty of missing words to fully enjoy.the ACE.
(*This work did not started for the computer itself. It started for what made it memorable*)
We try to share an attitude. As well as the lessons learned that may be useful everywhere.

How did all this started? (A personal experience, of delight. And then, of insisting questioning)
Got my Jupiter ACE after being sequestered by and to Military 'Service', as a result of changing from Medical school to Computer Engineering. (Both most unsatisfactory,'formal', bureaucratic)
Partly due the Jupiter-Ace, I would get my first job: To re-program the giant lamps placard built for the Oporto "Antas Stadium". This was the 80's, everything new. Even Orwell's "1984".

> *Years later, shared with the "Jupiter-ACE Resource" and its forums. It was interesting!*
> *Restarted investigating the ROM, started examining hardware details. By then receiving repeated promises of ..."Next month I'll contact Vickers"... "may GET the original".*

Never reached Vickers. Suddenly noticed to have a 'good' listing, supposed a disassembly.
This overlooked in favour of a legible real disassembly... Then re-learned Z80 ASM OPs.
Had ignored that gem at hand! When it was examined, it revealed itself by what it was.
We are thankful that output listing was saved. Even without its size reducing macros.
(Macros are a very personal know-how). The 'long' equivocation forced our "dive in".

> *Naturally, there was not 'one' original file. (Anyone who developed anything know this):*
> *There are sets of files. There are versions and their own sets, usually lost, over time.*
> *Glad those were lost. One of the reasons why (in spite of the loss) is easy to explain:*

<u>Absence of sources pushed us into other paths</u>, we describe as truly educative questioning.
This e-book is based on those sparse field notes. <u>Observations instead word of mouth</u>.

Here shown page-based (almost BLOCKS based). Each page having 57 lines (as you may notice).
Sparse notes are not easy to glue nor easy to edit. Then harder when connecting the dots.
Alternating between dissimilar sections did not help either (all was a bit too sparse).

> *<u>An exception was the benchmarks study:</u> Added because someone pushed 'net-official' lies.*
> *Enough to serve as (strong) motivation to get proofs, also get accurate and repeatable results. Now at the reach of everyone. (Must say we have learned "a bit too much".)*

Here, we try to reflect! To promote those days spirit, as seen on the Jupiter ACE adventure.
I.E. "to question" and remind the value of ingenuity. We ALL need to exercise THAT stamina.

> *<u>We All need Truth, Truth is all we'll ever be</u>. ("Yellow Submarine" soundtrack, please.)*
> *As everything in sight, not seen... We see IT as a parallel of the wordl (pun intended).*

A Few Chapter Highlights

- Ch 1.3 : An Introduction, with some of the missing words
- Ch 2.2 : Language and OS, small enough to fit
- Ch 3.1 : Important 'Tips' to Forth usage
- Ch 4.3 : A 'Primaries' Library (extended)
- Ch 5.4 : About ACE-ROM restoring (patches)

This Book-1 is not 'a' manual
It describes and extents ACE-Forth. Also delivers much searched Forth Internals (on Chapt X).
Appended Book-2 is different: It's plain Z80 code (ROM), plus my own Z80 OPs work-tables.

___
P.S. Much could be stated, useless to whom has not my own references. To each, their own.
Having seen too much, that much is not shareable. Then useless. Or with no reason to.
No reason to, because most would be taken as 'entertainment' (as fisherman stories).
Thus, different readers will read this e-book differently. Useful? We may hope.

How IT all happened

The Jupiter ACE attracted attention, due unexpected efficiencies and novelties it delivered.

Our first exposure to Forth come from an article mentioning a structured language reminding Pascal, with a 'strange' twist: It had to be used as Assembler does, with a stack. Informed it was FAST, very unlike the usual 'offer' (commands batch). I felt attracted and perplexed. Shortly after, the Jupiter Ace arrived. But so so little RAM! ... The Spectrum stated 48K, the Jupiter stated 3K. Later found to be less, just 1K. Much later, that 1k was equivalent to 3..4K ... The Spectrum seems (not sure) to be equivalent to Forth with 8..10K

In 1984 (savings lost) a Jupiter ACE was what I could pay (18.000Esc). A 16K pack was absurd as it doubled the cost: The Jupiter and 16K pack having the same price, costed 36.000Esc. while a Spectrum-48K costed ~33.000Esc. On that nonsense, destiny forced 'the' choice.

> *Though the Keyboard failed too much, the Flexibility of the Language was a surprise. Wondered HOW Forth Worked. Attempted to disassemble it.by hand. Byte by byte, using a Z80 Op grouping routine (got rubbish on Forth words) ... Examining the code had to wait*!

Years later, would peek the ROM again Using the only disassembly ever made (authors unknown). Examined the Jupiter Hardware (frankly, very crude). Then studied some didactic information (these are mentioned in the Thanks section). Not an easy process, but curiosity prevailed.

Disappointments: Found (too late) how the 'world' works behind 'our' tranquil assumptions. Over the years, noticed most memorabilia sites had other interests. Got evidence Wikipedia was less worried with facts, more on privileging 'fashion' (or worse), control a work-force. Experience first suggested (and then confirmed) an hidden "make-believe" gestalt:
* *Social push of Believing, as a centuries old replacement of Reality.*
* *Distortion to be wildly promoted by 'talk', easyness and 'format'.*

> *<u>1st, observation:</u> Assumptions (even plain wrong) were/are accepted as facts . 'Statements' shown incorrect and rectified, they do persist in that social play. (Happens on Physics, History, Medicine, Enforced Law.... And yes, Computing too.)*
> *<u>2nd, an example:</u> After wrong data imposed on Wikipedia, most shocking was the complacency from many (pretending to represent Jupiter Ace fans). Make-Believe 'rules' trough 'roles' (We now accept People to have very private interests, in the sense of very personal) Bof!*

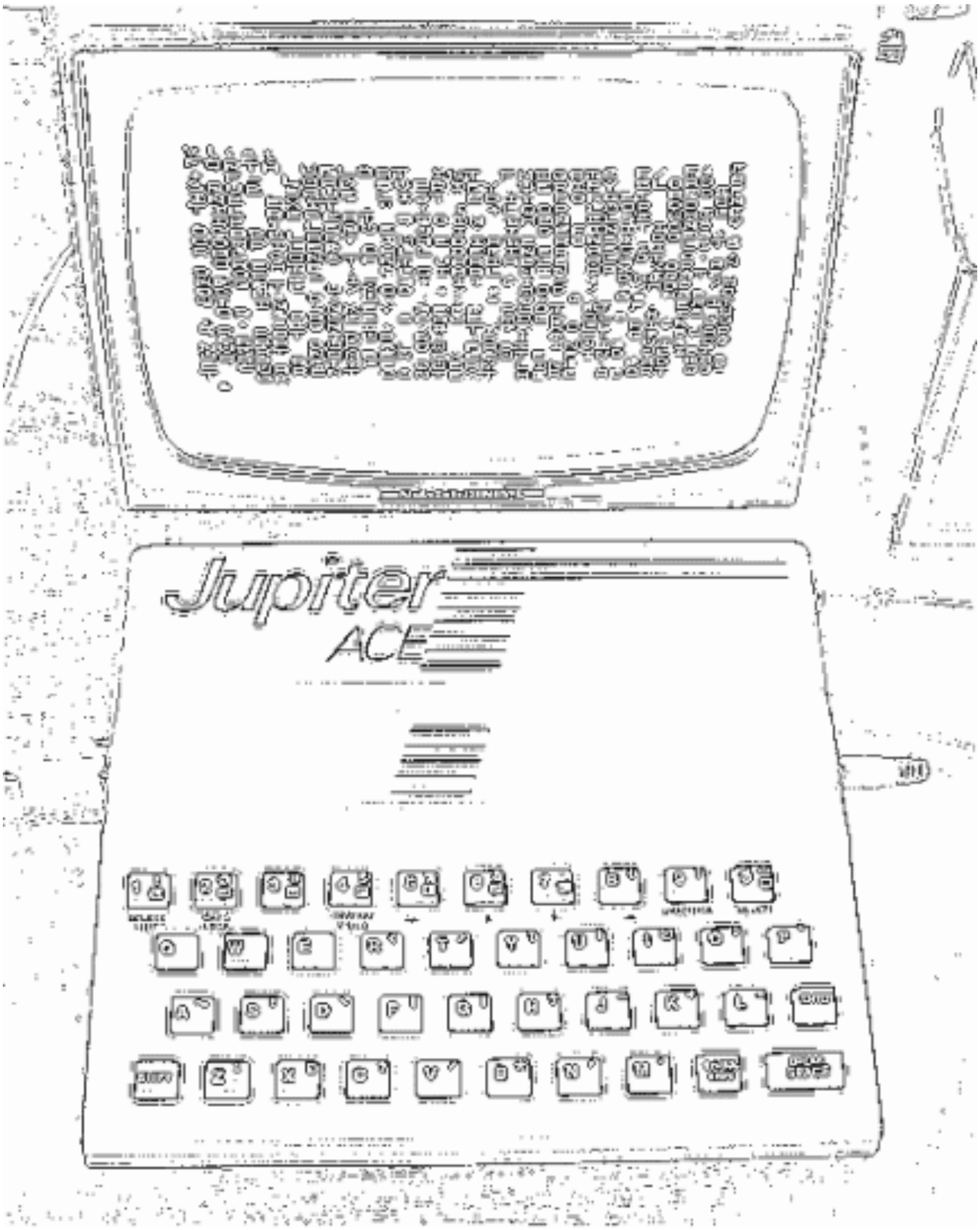
The Trigger: We felt compelled to correct wrong data, then sharing verifiable values. Confirmed <u>ACE-Forth</u> speed <u>WAS NOT</u> 10x BASIC, <u>BUT</u> **>15x**, as sensed (now everyone can check). *<u>Bad news</u>* is ACE-Forth, with a most surely desired rewrite... would show a ratio of **~35x**. Rewriting would also be a time saver: Easing development on the ACE's most tricky aspects. *<u>Good news</u>* is ACE vs ASM-Hand-Coding ratio is not /46 but /30 (words restored, down to /24). A fair result considering a Z80 and Soft-stack. Better than most CP/M compilers of the day.

Reasons for the initial SIEVE equivocations (on measuring a Forth) are:
1- SIEVE depends from a <u>limited set</u> of operations, not always representative.
2- When secondaries replace primary Words, delays (nesting and args) usage are multiplied.
<u>Both</u> reasons distort results (mostly the 2nd point above). Almost evident, both are overlooked. So, we demonstrated, built evidence. Still not enough,proved it with ROM patches. These ease checking. In the process (away from 'groups') restored a bit of the original intent... T<u>o ALL</u>.

In Parallel to this Documentation, everyone now can <u>confirm</u> results. Even check the methods. Conversion from Forth evaluation to Sieve has progressed by steps. You can find, on this free distribution archive, an individual Rom with one of our patches shared there:
> **ACE-TOOL** Rom (6% faster than the original) OVER is Primary. On 3.25 MHz it shows a Sieve ratio of **16x** Basic/3.5MHz. Now takes.**5min** (instead 6.5min) vs 80min of BASIC... Not much! SIEVE is available as code, and as easy loadable snapshots (for immediate verification).

___
 # We may wonder if Wikipedia controllers (and others) will ever learn the meaning of Reality.

Volume #1 - What, When, and HOW



The author's old 1K system. Learning and running ACE Forth, on 900 bytes.

= Volume 1 =
A Jupiter ACE Review

•

Details:
Document Date : 2022/June @LuxBonna, Lusitanea by the Sea.
Building Tools: An ultra-slim, lousy keyboard, small screen, dangerous touchpad.
Language used : International English, irrelevant form, weaved as a patches blanket.
This author learned to write and read a precise latin derived language
with plenty of Japanese (Nagasaki) and Indian influences (India, not Texas).

To whom it may regard: (or decipher)
The above are suppositions. Never opinions nor theories. Not even when joking.
To whom enjoys "to file", no longer with the care placed down “the rabbit's hole”:
As the 'shaker' would had said today: “To file or not to file. Ask not, roll the dice”.
Each paragraph is a place, with its own rules and consistence. Coherence is the pudding.

Legal Status, as enforced wherever GPL is applied or adapted:

Availability:
While now residing of https://t.me/JupiterAce the very last ACE-ROM_Doc_Prj edition can be found at https://drive.google.com/file/d/1ykjRsfCsfSOKw1YcOH6SyDjZT8hwUxtn/view?usp=sharing

A prelude quote:
When most I wink, then do mine eyes best see,
For all the day they view things unrespected;
But when I sleep, in dreams they look on thee,
And darkly bright, are bright in dark directed.

Then thou, whose shadow shadows doth make bright,
How would thy shadow’s form form happy show
To the clear day with thy much clearer light,
When to unseeing eyes thy shade shines so!

How would, I say, mine eyes be blessed made
By looking on thee in the living day,
When in dead night thy fair imperfect shade
Through heavy sleep on sightless eyes doth stay!

All days are nights to see till I see thee,
And nights bright days when dreams do show thee me.

~ (Quite True 43 years back, before this recall)

A few End-Of-Page taglines, to temper page reading:
To teach, is to point. Pointers available, not solutions.
To own is to do (understand)... Never tag or attribution.
Tags are just tags... Good and bad, all do have a chance.

Contents

BOOK I - WHAT, WHEN and HOW

A quick intro

A quick intro

Once I've experienced Forth, its capabilities become sources of amazement!

Forth was/is the smallest true language compiler ever produced. Practical and fast when most people only experienced batch commands engines, built to teach flow-charts (mimicking the lowest flow control).

Equally interesting, was the first and only detecting the need (on a CPU) for at least two stacks.
Why 2 stacks needed? To both simplify programming, as well as making it more efficient.
A lesson few CPU builders followed. First the M6809, later found on uControllers.

Casually, through the years, I've found pieces of information on Forth inner workings.
However, all were limited demonstrations on the HOW a particular implementation worked.
None worked as well as 1st contact, ACE-Forth. All become disappointing, so here we are.

Is it hard to say what I was searching? A simple, non-explanatory word, may help: "Essence".
To most people, that is just a word. In the void, it looks void. Let us contour its sense:
- *In spite a sense of the HOW things worked, "something was missing".*

I can say I wished to have by then, enough information available.
Anyway, personal experience can never be transmitted... Described, maybe.
Transmitted, no... It's personal. What we can do, is to share pointers. *We'll try.*

Chapter 1 - Introduction

“A small, beautiful, broken cathedral” ~ DuLac,2020
Why broken? It would shake the status-quo! (Who cares for greater benefits? Or country?)

The Jupiter ACE was an Home Computer still reminded only for its Internal Firmware: ACE-Forth.
It was designed and sold by Jupiter Cantab, a British company formed by (in alphabetic order)
Richard Altwasser and Stephen Vickers, previously labouring the Spectrum success (thus ignored).
While Jupiter hardware was flawed, firmware was revolutionary. With great values to remember.

Motivations were to build a useful Home Computer, i.e. really programmable, for more usages
than bought Assembler Games. This, by allowing efficient code.generation. Thus adequate for:

- Education ; Where a real language was needed
- Business ; Small business, needing flexible applications
- Industry ; Adaptable control tool with an accessible fast language

In short: An opportunity existed on delivering the very efficient FORTH (OS&Programming) as base.
Most Home-Computers in that market delivered a batch mimic of old FORTRAN, a cumbersome 'App'
meant to teach crude flow-chart design, or used as cheap human interface (mostly useless).
BASIC fit hobby low-specs: Was available in spite of code obscurity and “spaghetti code”.

Why to deliver FORTH? *(building one by then, was an art unknown)*
FORTH made sense, offered many benefits. However, it could break the status-quo.
(On the open, it was much welcomed. On the hidden, 'things' were a bit different.)
As a system and language, it had the right characteristics, was adequate on low hardware.
Could deliver efficient but manageable software (Structured&Compiled). Turned toys into PCS.

With ACE-Forth inside, was the first accessible computer with a real language built-in.
Not the more common scripted batch (useful but limited), nor compiled on Host machines.
No previous compiler ever had fit on 8-bit. All had to be hosted. This one was threaded, small:
The C.Moore threading system was not dependent of an Host for compilation. Had such a small footprint, while offering both good programming practices and the speed for practical work. Forth was truly Revolutionary. Delivered inside as a palatable Asm (not hiding the Stack).

Why to invest on a new system? Dragon Computers invited Steve Vickers to built a Dragon-Forth.
Dragon would be a winner over all. Then: *“Clive Sinclair was making all this money out of us, why shouldn't we just make our own computer and make the money for ourselves?” ~ Steve Vickers*
A favourite quote on the dream of merit. (Opportunistic investors, gone Machiavelic, want all.)

FORTH allowed and justified it! Problem was, FORTH should be adapted to common tape-Recorders.
FORTH already managed Tape-drives, opposite to Recorders and very similar to Disk drives .
The ACE made the adaptation to Recorders, thanks to Decompiling. An hard job, unique.

Vickers was a Software-Wozniak, Altwasser was an Hardware-Sinclair. (And the secret investor?)
The incomplete machine evolved differently. Result is ‘history’... of dreams "that could".
As an informal joke, Forth on 1981 was as OOP in the 90's (I.E. as SEX by the 50's):
- Everyone talked about it, no one knew how it worked. (also reported by Loeliger)
As today, ‘Experts’ on mirages popped, all exercising impressive 'presentations'.
By then it was *very* hard, to get a good insight instead ‘ideas’. Knowledge was mostly work,
over trust, help needed. (Today we face the opposite: An "information flood". Too much talk.)

___
 # "Not possible" can be a great teaser. Be it an algorithm, an architecture... whatever.

1.1 General Description

- ► Testimonies & Quotes
- ► A crude description
- ► Driving Architecture
- ► Decisions, decisions

On a beautifully designed stepped white case, showing contrasting black rubber keys with resistive rubber contacts. It displayed a monochrome output onto an available home TV. Being an accessible (inexpensive) computer, programs were to be saved (and loaded) into an available cassette tape. Common choices to avoid expensive diskette drives. So far, nothing new... However, it was unique:

It's major appeals were both a real language, was 15x faster than its Z80 based competition. (Relative speed was incorrectly measured as 12.4x, rounded to 10x (see ACE_ROM/Patches&Changes) A crude view, as there was more than speed. It had a real Language, finally available to ALL.

Forth offered a long waited solution to an old problem: A man/machine interface, not a batch. Nor 500x slower than ASM as BASIC. Problem was, a compiled language using cassette tapes for storage, needed “something else”. So, ACE-Forth reconstructed sources from its compiled code.

Decompiling allowed to edit programs with no sources from a disk. 'Redefining' was also added:

- *"So one of things I thought was going to be necessary was an in-place editor, which is different from the standard Forth model." ~ Vickers.*

To achieve direct decompiling (direct access to code) a few changes where needed:

- *"There are novel features in my Forth." ~ Vickers*

Hardware Body -- The Jupiter board mainly consists of:
CPU (Z80) ROM (8K) User RAM (1K) Video RAM (2K)
On the board most chips are for the video board:
Screen buffer(1K) plus Chars bitmap(1K) ... (Should be one 4K chip)
Video-logic circuits driven by the clock crystal (shared with CPU)
A small metal box for TV video output (an TV UHF-Signal generator)

((To keep it afordable, it had only one video mode of 8x8 user-redefinable chars. Video displayed 32 columns by 24 rows of black and white 8x8 tiles (characters). The tuner box was needed to feed a TV, not yet with direct signal connectors. To use a TV, a computer had to put the video signal onto a TV carrier, usually a UHF channel (frequency). The user selected that computer broadcast on the TV. The tuner box was there to translate a screen dot signal, into a TV UHF signal the TV would accept (once tuned), then show the broadcast dot on its screen.))

Video section was what the ZX-80 should had (against engineering wisdom, the ZX had not). It allowed to display graphic chars, by redefining the 8x8 pixel bitmap of any of character. Free 22 chars were available to be redefined, 16x16 sprites with 4 Chars. There was much more:

Firmware Soul, the real deal, Was the major difference in face of all 'introductory computer': The Jupiter ACE was idealised as a support to good programming practices and fast execution. These were needed (instead lined scripts) by Entusiasts, Students, Programmers and Industry. No longer a batch for exploration, but a self-compiled language, structured (the real deal)

In short:
It offered FORTH, a fast, well structured compiled language. This language was extensible (adaptable) using either Forth routines (Secondary Words) or ASM routines (Primary Words). Planned to have 4K RAM (equivalent to 12..16K). Maybe color as an option.

 # "Utinam tam facile vera invenire possem quam falsa convincere" ~ Cicero, De Natura Deorum 1-21

Testimonies & Quotes

One testimony:

I loved my ACE. I wrote software to demonstrate stuff in my physics A level class.
I also wrote a program that let me whistle Morse code from the other side
of my bedroom and it would interpret it as text.

*What I liked about it was the way you could REDEFINE words *without* having to FORGET*
everything defined afterwards. Do any other FORTHs work like that?
~ Anthony Hegedus

Another testimony:

I loved my ACE. At that time, to me, it was a mystery (how did it ticked so fast?).
*One could REDEFINE words *without* having to FORGET everything defined afterwards.*
Much later, I've found that to be unusual... It seemed very natural.
Unfortunately I've only enjoyed 1k (actually 900 bytes).

Also become aware that, from the 5K available to FORTH,
nearly 1.2K was occupied by Dictionary Headers. These were needed, but
showing ACE-Forth code actually occupied ~3.8K (including EDIT, TAPE and FLOATS).

Now, I know it did not had to be that way. Fair examination shows that was just
the result of unpaired development of the hardware in face of the software
(This is harsh thing we cannot say, because painful no mater how true).
However, not even the love for a son should ever make us blind.

After all these years, passed, still admire it.
Even more: In its beauty, and with its flaws.
~ Dutra de Lacerda

A quote from a legend:

"It is practically impossible to teach good programming
to students that have had a prior exposure to BASIC:
As potential programmers they are mentally mutilated
beyond hope of regeneration."
~ Edsger W. Dijkstra

About Forth:

((Its script advantages while not being a script, for compilation as for teaching.))
"Forth is entirely interactive. You type in one word and the system automatically looks it up
in a dictionary of definitions and then executes it. The user can define new ones,
and programming is like writing subroutines."
~ Stephen Vickers

About ACE-FORTH:

"May be the easiest introduction to FORTH that ever existed.
Maybe too good, as it offers some utilities Forth itself lacks.
Not just the simple Editor directly available, but WORD Redefining.
Reasons are plenty. After starting one already feels at ease with FORTH."
~ Jupiter ACE ROM Documentation Project

A crude description

Jupiter Ace Summary

- OS : ACE-FORTH, a Forth-79 language subset, with some changes and Floats.
  Fast enough to allow a wider screen (lower freq = bigger screen area).
- CPU: Z80A running at 3.25 MHz (so to get a wider, more natural screen view)
- ROM: 2x4kB EPROMs containing BIOS system routines, FORTH compiler and editor.
- RAM: 1k [originally 4K, only got 1K) Z80-bus on the rear allowed 48k expansion.

- Video Display: 2K [dedicated Video, not the ZX-80 expensive hack]
  - 32 x 24 Monochrome Chars (128 definable tiles) software built,
    initialised with a local ANSI (font British typewriters based)
  - 64 x 48 Plotting (using Mosaic Chars)
  - 3 input modes: Text or Graphics (Mosaics) x INVERSE VIDEO

- Keyboard: 40 rubber keys (directly conductive), used with auto-repeat.
  (two shift keys allow all ASCII codes to be introduced).
- Sound: Single channel buzzer (CPU driven).

- Program Storage:
  - High speed 1500 baud to/from an standard cassette tape (modem like).
  - Dedicated full LOAD/SAVE 1500 baud routines (for the protocol), plus
  - Localised VSAVE/VLOAD routines for specific addresses and block size.

- Interfacing:
  - Power (9v), stabilised to 5V.
  - TV connector [UHF TV set to Channel 36]
  - Cassette port x2: Ear & Mic
  - 2 edge connectors for expansion:
    1) has a complete address and data lines from the CPU,
    2) selection lines to Screen Data Array (1K) (not practical)
       and A/V signal much used by home clones on later SCART TVs

Physical data (nothing to report, irrelevant in face of a picture)
- Weight: 246g. • Size: 215 x 190 x 30 mm. • Documentation: 182 page Users manual.

Notes

Using a real language gave it many great advantages (over "interpreted BASIC being used on most 8 bit computers). Yet, its sound and graphics capabilities unnecessarily compared poorly in face to upcoming competition. Too many circumstances usually overlooked, pushed this computer into a niche market... Sadly away from what it could had offer to England, Europe, to all.

Also to mention that school teachers where afraid of a real language (specially WHEN new to then). Academic wannabes, which are very common, seen climbing to 'positions', despised 8 bit machines though fascinated by the attention and by the number$ at play.

Home users favoured commercial games (in ASM): Few users did more than to copy hard to read (BASIC) listings. Enthusiasm was covered (not replaced) by a new trend. Worse was the warning that it had only 3k (1K+video, equ to 4k on Basic) or that it run at 1MHz (as a “What Computer to Buy” booklet, a magazine edition, stated).

As a result, Sales were never very large. Reported Ace's sold before Jupiter Cantab closed was around just 8,000. Surviving units being rare, curiosities for collectors, or an “I have one”.
... Its insides being lessons in several areas. Dead Jupiter, running ACE.

FORTH is much used in uControllers (when host programming is to be avoided)
BASIC only remains as a name, quickly replaced by... morphed Pascal and C engines.
Also ‘morphed’ was the honourable BBC BASIC, with a much different (threading?) system.

Functional Blocks

```
$0000+=====================================================+     (addr)    +=====+
              \                                                /   ---bits--   |     |
               \                   ROM 8K                      /  A10 A11 A12  | ROM |
                \                                              /   v   v   v   |     |
         $2000+=====================================================+ +---*---*---+ *=====*
              |       echo of $2400 (Direct)       |0|<  cassette  >| |   |   |   | |     |
         $2400+ VIDEO -----------------------------+-+$2300---------+ |- -| 2 |   | |     |
              |       SCREEN BUFFER 768 bytes      |0| PAD 254 bytes| | 1 |   |   | |     |
         $2800+------------------------------------+-+$2700---------+ +---*---* 4 | |VIDEO|
              |       echo of $2C00 (Hardware)                      | |   |   |   | |     |
         $2C00+ GRAPH ----------------------------------------------+ |- -| 2 |   | |     |
              |       GRAPHs BUFFER - Write-Only (dedicated RAM)    | | 1 |   |   | |     |
         $3000+=====================================================+ |---*---*---| *=====+
              |              Absent, $3C00 Answers (echo)           | |   |   |   | |     |
         $3400+       ----------------------------------------------+ |- -| 2 |   | |     |
              |              Absent, $3C00 Answers (echo)           | |   |   |   | |     |
         $3800+       ----------------------------------------------+ |- -|- -| 4 | | RAM |
              |              Absent, $3C00 Answers (echo)           | |   |   |   | |     |
         $3C00+-------+---------------------------------------------+ |- -| 2 |   | |     |
              |SYSVARS| DICT {12} DATA STACK ->        <- RET STACK | | 1 |   |   | |     |
         $4000+=======+=============================================+ +- -+- -+---+ *=====*
                             48K AVAILABLE for expansion              \   RAM   /     +
               (Note Ret Stack is always at the end of available Ram)   \ BANKS /    PACKS
         $FFFF+=====================================================+   (sizes)    +=====+
```

2x4K ROM, 2x1K Video and 1x1K SRAM (1K SRAM as expensive as 4K DRAM)
    'Echos' result from absent lines (no 4K SRAM internal multiplexing)
The two 1K SRAM (static) Video Buffers are for Screen Buffer and Chars Buffer.
    Each byte echoed in two addresses (bit A10 was not de-multiplexed).
Three Blocks of 16K Addressing to accept RAM expansions, or two blocks and a system area not used.
RAM expansion come as a pack of 4x4K dynamic RAM (8K made no sense, the 'pack' was the main cost).

User memory   : Designed to be 4K (ROM size allowed it) become 1K (forcing an add-on).
Video Memory : Memory Map suggested Address space to Color ona 2k Video Buffer (->2K), and/or
                a socket for a Graphics Buffer extra-bank (->2K). Inv-Video closed that chance.
BIOS ROM   : Inexistent. Placed on FORTH ROM space, stealing 3K (space down to 5K, rounding).
Expansion ROM : Inexistent. We may assume initial projection to be 2+6K, expansion was advisable.

Analysis: 4K wide was the default RAM Bank space chosen, the minimum to expect. Equivalent to ~12K on a Tokenized Basic. Therefore, a fair starting base. Strangely, 4K DRAM was neither implemented, nor allowed. The projected 4K (a tweak on initialisation code would adapt to both ‘bases’.
Board add-ons are MUCH more expensive than chips (add board, connectors, mounting, distribution).
A too old place for ROM and the absence of BIOS ROM, are major flaws of this anachronic design).

With near 900 bytes of RAM available for programming, it was still enough to exercise the principles of real programming (with a real language) and do some real work too!
But hardly! Learning with it, soon fighting with the the limitations then found.

Conclusion: With 4K User RAM absent and no internal sockets for expansion, users where forced to a double investment: Computer + RamPack (this no longer an option, disregarding initial stated goals).
There was much enthusiasm, forgiving much for a while. The 1K frustration would rise (though later).

- Only its FORTH delivered and surpassed its promises. Even if 'reduced' (See Ch 5.1 What's on Rom) It's for that "much more" of the ACE, in so much less, that the Jupiter ACE is honoured.
- We conclude the initial ideas for the Jupiter were dropped, in favour of the 101 exercise exhibited by its circuitry (only adding the dedicated Video that older ZX-80 should had).

Being fair, with those frustrations the compound also gave us rewards and opportunities. We kept it!

Face and Keyboard

The author's own 1K system : 2 Shift Keys and 3 Char modes(Caps,Inversed, Graphs)

A beautiful layout design... After a significant logo (Forth/User) or (CFA/words, ‘;’)
The architect, designer, sculptor, marketer, whoever (one or many) made a beautiful case.
Even if later built uniquely weak. Never seen a box that miserable, nor without a gain.

A most uncomfortable keyboard... Final touch. Of discredit. (by whom? the secret investor?)
Keys were also uniquely built too high, bouncing to a corner. On an also too big movement.
Even if it built without experience or competence, it should be tested. Why uniquely high?
After test, correction would follow. Was it not evaluated before built? Nor by the client?
The too big key movement is amplified by Keys too high built uncomfortable, prone to fail.

The worst rubber keys EVER made for the most fragile box ever used, and expensive 1K SRAM.
We are not just surprised by absurdly huge Keys (almost cubes), made to bounce.
Aggravated by a too long distance to contact. Both ensure input failures.
Why so many wrongs from ‘expert’ factories? ... Then also questioning the Motherboard:
Hand-designed board failed the 4K RAM space, even bloked later correction chances..

___
 # Strange (anti) sports are played in the rotten City of Denmark... (city, must be).
The Kali rule states for where it nests: *To the best, the worse. To the worst, the best.*

Decisions, decisions

Back to 1981

Time, RAM prices and marketing demands seem to be the reason why 1K was used. Not true.
The absence of colour may be justified with Time and money being short. A parcial sense.
Was that all? The little beast would be a game changer, shaking the status-quo.
The usage of (commonly used) 4K dynamic RAM, or a socket... was necessary!

Reducing chips and options when 16K becoming standard, when more factories producing cheap
4k DRAMs turned standard long before... That, makes no sense. Everyone had 4K DRAM chips.
The marked was growing for 4K DRAMs. It was needed, as was a socket for expansion.

Also: BIOS also meant a place for Tables (ANSI chars and a few other), ie, their own ROM.
Common Z80 Forths' where known to fit on 6K (not including BIOS). These where 4K + 2K dictionary
(very small hacked Forths', fit on 3+1K). Development also demanded a similarity to the developing
CP/M system (the most natural to be used). Specially when firmware was the hardest task.

The CP/M example showed the a way to build a better Z80 machine, an hardware design not followed.
A BIOS away (as usual) would free the ROM to the usual 6K FORTH. Yet, the blueprint was limited
to the test breadboard ...As absences do show. Of no alternatives after against initial goals.
(See next section: System Architecture)

May had played a role (we see no sense there) hopes that a 1K ACE would be:
1 - An entry bought (with a forced expansion, opposite to goals?!?)
2 - Enough for learning (Buy, experiment, learn, say good-bye)
3 - A cheap controller to industry (no safe with an expansion)

Expansions were/are more expensive than a chip inclusion. Expansions violated initial goals.
Even if half-correct, previous suppositions -1- and -3- were flawed:
1 - Other machines were coming, not demanding addons (and offering colour).
3 - Connecting a RAM add-on was a no-no to industry demanding reliable (but cheap) components:
Demanded socked expansions, independence, modular systems.

Never happened...

For its initial characteristics, it would be a game changer. So it did not happened.
We may guess it must have worried several people. Very likely... Maybe (%).

The Video system was never modular (would allow an alternative Colour Block).
The ROM was forced inside the 8K ROM that would be reserved to Forth.
The RAM was reduced to 1K static, of the same price of 4K dynamic.
The keyboard was made bouncy, and the box uniquely fragile.

Had Jupiter Cantab spent a month, their hardware could be properly designed (more like CP/M).
Even get colour without a new board (a simple phase delay requiring very little on dedicated video).
Tests would be followed by a 2nd phase, with DRAM. The blueprint suggests it did not.

Had Jupiter Cantab spent a week, their firmware could get a few more bytes (Sieve ~1.5x faster).
Not fully correct: With a month, a rewrite of ACE FORTH with CPUstk as DataStk would be **~3x faster**.
Surely noticed and desired. Development would be easier, then recovering more than the month spent.
Why... was that not done when all evidence shows ingenuity and care? (See Ch X, Other 'solutions')

Had Jupiter Cantab spent a month, hand-designed board would had internal sockets (a common practice)
present or later soldered or (or allow use of cheaper 4K SRAM chips). Instead, it's what we may see.
Had the ACE those initially mapped 4K... then History could had be different.

Time and money were short and it's hard to predict the future... Yes.
The opportunity existed, more or less clear. A month well spent would be not lost. Was not.
Britain would benefit, everyone would. We all would... But it could had been a game-changer.

Questions not asked

[MHz and Video Areas]
The reason the Jupiter Cantab computer run at 3.25MHz and the Sinclair series had 3.5MHz,
while others run at a quicker 4MHz was ...the Video image size!

To reduce circuits, Video and System were co-dependent (a needed simplification).
As such, 4MHz would mean shorter dots (= too small visual area). A compromise between
the faster MHz (for the usual reasons) and the smaller area people could accept (for comfort).

If compromise was 3.5MHz, image would be very small. As this Jupiter Cantab computer was very
fast due FORTH, it needed no such compromise. Therefore, a wider visual area was allowed
(borders of 4 chars, 3 lines on top, 3 on bottom) running at a slightly lower MHz.
We may say it needed to look 'natural' with short borders, as it lacked color.
... Thus, 3.25MHz follow the (initial) easy to use goal. Now you know.

[Limited by design?]
RAN production rise with demand. Dynamic RAM were available but ... trough resellers.
Requests had a pair of weeks delay. Only. Dynamic RAMs were cheap, 4K RAM was needed
Yet, no upgrade was printed, not even as connections (for soldering absent sockets).
As common practice they should when considering the handicap. That has bothered us.

Even so, an expensive MUX was kept in place of a pair of NANDs. Reflecting breadboard tests only.
Not a suicidal decision (if ever there was one). Without preparatory work, little was achieved.
Then, a pack would add up to 50% of the Jupiter price (100% for mine) suddenly not making sense.
When I got my Jupiter, after 1983, a pack cost the same as a Jupiter-ACE, 16.000Esc.
A Spectrum 48K was 33.000Esc. Naturally, refused the pack. And felt the 1K limit

That RAM miss FORCED people to later buy a 16k pack (nearly 10x individual chips prices,
40x when considering the initially projected single 4K onboard instead 16K). Consider this:
Even more expensive 4x 1K static RAMs would be cheaper than 1x 4K pack, IFF on the blueprint.

Worse, an extra 4K system ROM was needed not to steal 2-3K from FORTH. (Check What's-on-ROM.)
The chance of evolution was NOT implemented (we guess the projection was 6+2 would fit).

[In retrospective]
While the Jupiter was very little, the ACE was a feat. Still admired today as its true, real value.
Even after constricted into 5k (delivering the equivalent of a 3.5K Forth). Then hardly helpful,
with 900 bytes RAM to run. (With emulators and its available RAM, we do not notice the trouble.)

Was a feat on and due the tape-recorder era. It needed disassembly. Never done before nor later,
not the same way. The tape-recorder era would last a few more years until Diskette drives become
affordable (later disk drives, now Solid-State). Disassembly and Edit inflated as ~2K extra(!).

Though ACE-Forth was a feat, we could speculate that failing the extra space (also demanded by
the Floats library) may have been a consequence of bad communication (maybe after 'investors').
Reflected and aggravated by the unpaired development of hardware and firmware:
* The ACE become too good, the Jupiter become much, much less.

So we need to question: Who managed Jupiter Cantab, answering to the investors?
How much did they interfered? More importantly: Who where they, really?
* Others, in the past, without ingenuity had ‘managed’ engineering:
  Either lucky (the ZX), either 'blocking' their own (M-6809).

What remains from the Jupiter ACE... is the ACE, even if not complete. *(Now it is)*

___
P.S.: We tried to return the joys, and the learning we have experienced those days. We can do that
with small contributions. Our Library, with what we missed. Opening doors, searching the WHY.
These last are due too... As we remind every single day, "*The truth, is all we'll ever be*"

1.2 Jupiter Hardware

- ► A Few Notes
- ► Memory MAP
- ► WORK Area
- ► Things *not done*

**** Base Sketch ****

64K Memory Addresses as 4x16K Blocks

System uses first of four 16k block. This is 8K ROM and 8K Work Areas.
3K BIOS shares ROM with 4K Forth+1K Edit/Disassembly (3+5 shares)

Work Area has 4k space to Video and 4k space to RAM.
Only 1K is used, without option to use 4K dynamic RAM.
Test Breadboard was implemented (board print), not the original design.

Surprisingly, is not later_sockets_ready. Patching is only possible with cutter and wires. IE:
NO options exist for MODs beyond the Expansion system: No spare space, nor sockets.
It's limited and final, as many other 8bits 'constructions'.

```
 Address          Address        Address       Address
             BITs     16K     Bit            Bit           Bit
           A15 A14   Blocks   A13            A12           A11
            v   v      V       v              v             v
          +---+---+----------*---*----------*---*         *---*
          |   |   |          |   |          |   |         |   |
          |   |   |          | 0 |   ROM    | - |         |   |
          |   |   |          |   |          |   |         |   |
          |   | 0 |  System  |---+----------*---*---------+---+ ----
          |   |   |          |   |          | 0 |  Video  +---+ See
          |   |   |          | 1 |   Work   |---+---------+---+ WORK
          |   |   |          |   |          | 1 |   Data  +---+ Area
          | 0 |===+==========+===+==========*===*=========*===* ----
          |   |   |          |   |   16K    |   |         |   |
          |   | 1 |   Free   | x |   RAM    | x |         |   |
          |   |   |          |   |   Pack   |   |         |   |
          |---+---+   ---    +---+   ---    |---|---------|---|
          |   |   |          |   |          |   |         |   |
          |   | 0 |   Free   | x |          | x |         |   |
          |   |   |          |   |          |   |         |   |
          | 1 |---+   ---    +---+   Free   |---|         |   |
          |   |   |          |   |          |   |         |   |
          |   | 1 |   Free   | x |          | x |         |   |
          |   |   |          |   |          |   |         |   |
          *---*---*----------*---*----------*---*---------*---*
            ^   ^              ^              ^             ^
           32k 16k            8k             4k            2k
```

The main back-slot

A simple direct access to the Z80 bus (Z80 bus socket).
It SHOULD have been compatible with existing hardware add-ons.
Z80 bus not being copyrighted, neither signal positions. It would be abusive.
Add-On limitations for "anti-competition", as cartel agreements, would (?) be illegal.
Lets not be naive. We have seen the execution of law being used against it.
Main examples being 'bogus' law-suits, using 'law' against victims, to destroy or submit.
(As the famous pseudo-case against Digital Research. It would bankrupt under legal fees.)
(Or more recent ones, as seen with the great civilisation slide out of the last decades.)

A Few Notes

3K Total, or 1K USER RAM ?
It is common to refer the Aces Memory as 3K, by the 80's practices. This refers to the total of its RAM by adding the 2K of the Video. Though it must be also stated the ACE actually could do more with 1K (actually ~900 bytes) than any BASIC with 3x more RAM. We guess because cleaner.

User space was reduced from initially designed 4K to 1K USER SPACE, making the ACE marketing comparable to the ZX-81. Such equivalence was damaging PER SE, almost as much as having a quarter on the initially projected RAM (that would be equivalent to ~14K on BASIC usage, BTW):

A 4K Design... ignored or discarded?
The ACE was designed around 4K regions. Both for Video (half cost), also for User RAM (usability). Would doubled tiles to 256 tiles, thus better graphics. Colour (just a time delay) needed 2 Pages.
Thus, Video was reduced from 4K(2+2) to 2K(1+1). Trying to save a chip or two? And yet:
A 'formal' MUX was kept, instead a cheaper NAND... Used static RAM instead dynamic RAM.

As only 1K was used, ROM Code was compiled pointing the last 1K, so to continue with add-on RAM. This also inhibited any upgrade from 1k to 4K: The DICT will not benefit To do so, the whole system would change with loss of backwards compatibility. A 4K Base now means a new system.

COLOR READY (?!?)
Colour is not that hard, nor that expensive. It's just a precise (on-colour) delay.
The way the ACE Address Space was designed, colour 'would' be natural, expansion or a base.
That would not be a priority, possibly due goals and timing. The board would only need a socket for attractive open options. Nor even connections to solder one. Compatibility would not be lost.
Instead, an inverse video (instead half cost RAM) invalidated many chances... What was the sense?

Colour would work similarly to the VIC, without a need to buying the VIC chips (nor using a ULA):
* *Separate Colour Video-Buffer (at $2000 1K bank) so each char had its own colour attributes.*
* *Colours could use 2 x 4bits RAM, each byte corresponding to Chars in the Video-Buffer.*

This would give 16 colours to the Background and another 16 colours to the Foreground. Or 4+4.

BIOS ROM (where init tables)
All this and more would demand an extra ROM (traditionally at $FC00). Hardware must play safe.
It would free the FORTH system (consult Chapt 5.1) after enhanced with a ~1K extra utilities needed on its first record incarnation (and ~0.5K for disassembly). An all new environment!
Then could be a complete 6K Forth system (plus extensions). Never assume, play safe.
4K Forth, up to 6K, all rely on external BIOS. Was it supposed to be 2K? Even so.
It's amazing this ACE (+ ~1.5K 'innovations') fit on this Jupiter ("post-goals").

NOT SO FAST! (And yet...)
It's easy to talk from distance, evaluating with surplus time. But design was for 4K RAM.
Plus 4K ‘safe’ ROM. The BIOS (on CP/M days) at the end of space (not to freeze Z80 RSTs).

This ignored as if without alternatives. In contrast, we see many innovations (only) available on the ACE. <u>The ACE grew, while the Jupiter Hardware design shrinked</u>. Burdened with the lack of a system ROM elsewhere (surely considered: CP/M days). These are not light observations from an comfortable and far observation point. These were critical.decisions.
(Strange things do happen, weirder than these inconsistencies observed.)

What does shock us most, are the steady set of troubles (concurrent, coherent, and recurrent) regardless of the hardware design. Itself a series of misses and replacements: inverse mode to replace 128+128 tiles option, BIOS. Then, problems seen that should not had EVER existed (but imposed) as the Keyboard built to be bumpy, a Case to discredit, add to the mentioned.

All troubles downgrading a computer that would fly, favour customers and students. And England.
So we look further, wonder how everything works... Beyond the 'show' and ‘thrown' statements.

Memory MAP

MEMORY MAP (from the English disassembly listing, edited)

```
$0000#=========0============================================#               #=====# -----
     + $0400 --1--                                    2k ---+               |     |
$0800+---------2----------------- 4K ROM -------------------+               |     |
     + $0C00 --3--                                    2k ---+               |     |
$1000#=========4============================================#   / Addrs \   # ROM # --R--8K
     + $1400 --5--                                    2k ---+  /---bit---\  |     |
$1800+---------6----------------- 4K ROM -------------------+  A10 A11 A12  |     |
     + $1C00 --7--                                    2k ---+   v   v   v   |     |
$2000#=========8============================================# *---*---*---* #=====# -----
     |       (fast access)       768 bytes |0| < cassette > | | 0 |   |   | |     |
$2400=SCREEN BUFFER (1K) ------------------+-+--------------| | - | 0 |   | |     | -r+w 2K
     |       (safe access)       768 bytes |0| PAD 254 bytes| | 1 |   |   | |     |
$2800+-------------------------------------+-+--------------+ +---+---| 0 | |VIDEO| -----
     |       (fast access)   Realtime VideoHardware access  | | 0 |   |   | |     |
$2C00=TILESs BUFFER (1K) --- (8x8 Chars Definitions) -------| | - | 1 |   | |     | --W--2K
     |       (safe access)   Too slow -> just define Chars  | | 1 |   |   | |     |
$3000#======================================================# *---*---*---* #=====# -----
     |                                                      | | ? |   |   | |  a  |
$3400+ - - - -         Wasted Address Space         - - - - | |   | ? |   | |  b  |
     |                !! Ready to be used !!                | | ? |   |   | |  s  |
$3800+ - - - -          ( the 1K ram MUST )         - - - - + |   |---| 1 | |  e  | -r+w-4K
     |                  (   be removed    )                 | | ? |   |   | |  n  |
$3C00+ - - - +--------------------------------------------- | |---| ? |   | |__t__|
     |SYSVARS| ...DICT... () DataSTACK->        <- RetSTACK | | ? |   |   | | 1K  |
$4000#======================================================# *---+---*---* #=====# -----
     :                                                      : :    16k    : : RAM :
     :             48K AVAILABLE FOR EXPANSION              : :    16k    : : EXP : (r+w)
     :                                                      : :    16k    : : EXP :
$FFFF:======================================================: :-----------: :=====: -----
```

The ACE uses the first 16K Address Space as follows:
System = 8K Space is Sys+Forth ROM (3K+5K)
WORK = 8K Space is WORK space with 2 areas
** 4K Area of the VIDEO sub-system (2+1K, instead 1x4K)*
** 4K Area as USER space (1x1K, instead 1x4K)*
Following 16K free for USER space Expansions. Made needed against initial goals.
Next 32K for User or System Expansion (RAM or Hardware)

Notes:

- On the 4K WORK Block, the existing 1K is considered to be at the end of the first 16K block ($3C00 - $3FFF) to allow continuity with expansion RAM-Packs.

- Calls to any of this 4K addresses will be answered by the 1K present. It ignores address bits 11 and 10 of the 4k it replaces (echoes them). So, for the sake of continuity, code references it starting at addresses $3C00 up to $3FFF instead 4K starting at $3C00 (which is the echo? All!)

- Similar 'echos' happen on Video areas, difference being signal A10 (address) determines priority of access, only used not to disturb Tape access timings. (Should remind here that a latch would be preferable, its usage with 1x 4K dynamic RAM meaning half the cost than 2x 1K static RAM.

- (Maybe not a stupid choice as it seems, maybe just anachronic: ... It made sense at a time when dynamic RAM was not an option. So, and again, we need to question: Is this a much older design?

A simple Mod, for clones

After-the-Fact

For clones with 4K RAM on User Space, this is not (ever) claimed by ROM.
Altering the ROM for 4K RAM needed in the 80's is now theoretical, has no use.
But it is an interesting exercise, for a class room, and teaches to open perspectives.
(As long as seen for what it values. Not pretending to be something else, as seen everywhere)

The usage of 1K limits the previous ‘design’... What can be done?
We could use it for both hardware expansion and/or porting options.

On emulators or on a clone, we can use the 1K flaw making 3K available as safe location for Asm routines (kind of small ROM expansion slot). OR as a TEMP work area and RAM buffers.
Here's an example of usages for a full 4K User Area (available without ‘echoes’):

```
_______________ Address BITs __________________         Eventual Options to
           A15  A14  A13            A12   A11   A10         this Data Cluster:
           32K  16K   8K            4K    2K    1K
          *---------+---+--      ------*===* *===* *===*---      =============== $3000
          :         |   |0 is   /      |   | |   | | 0 |---      2K free for
          :         |   |Video /       |   | | 0 | |---|---      ROM Expansion,
          :         |   |     /   User |   | |   | | 1 |---      or ‘island’ RAM
          :  0   0  | 1 |-----    Data | 1 | +---+ |---|---      =============== $3800
          :         |   |1 is     Area |   | |   | | 0 |---      1K Buffer/Pad
          :         |   |User          |   | | 1 | |---|---      =============== $3C00
          :         |   |              |   | |   | | 1 |---      1K User RAM
          :         +---+--------------*===* *===* *===*---      ===============
          $4000
```

Desiring to keep compatibility (after the fact) the unused 3K can have any function.

To solve the echo problem, the User 1K RAM present **MUST be relocated OUTSIDE the Jupiter ACE** to use the desired 4K without conflict. Even then, only 1K will be immediately available to Forth.
To use the added Ram, or Rom, access Words may be used (to hide details).

On this trivial example, as the supposed ‘design’ leaves little chances: ROM rewrite, or Buffers.
One can be used for disk access (a disk sector has 512 bytes. The other can be used for other purposes, split in several smaller buffers for system and program usage (we rather have 4K).

Due to it's nature, it SHOULD be reserved for hardware-only purposes (by extensions to the ACE).
It is surprising the most successful clones do not implement the full 4K were it belongs.
Not even emulators (we refuse to build one). Nor a ROM rewrite (easy but useless).
So: We do not see it happen.

We do see (4 decades later) something similar to “broken glass syndrome” affecting the JA.
Because the Jupiter was not. Even after all this time (opposed to deserved ACE enthusiasm).
Emotions once placed against the JA, for exposing the delusion (profitable and damaging).

Will not go pointing the continuing attempts (there has been many)... We do the opposite:
- Try to clarify, share enough so anyone can verify the realities of the delightful spirit behind the JA, cause of its demise. We do suspect the hidden investor, obtuse, protecting the status-quo without vision. Against later benefits.

This too, confirming centuries old wisdom: The deceived protect their deceivers. We can all confirm that much, everyday and everywhere. Where satisfaction grows without merit, just luck.
Questions is: WHY do the best intentions fail? The How, we know: The distress of luck alone.
We might say that distress to be a profound misery that tries to cover itself. Desperately.

WORK Area

System Block

```
- ROM is 8K    -> [A15,A14,A13]=0
    System ROM does not exist, stealing the above 8K by ~3K.
- Video gets 4k space -> (1k Screen Buffer + 1k Pixels Buffer)
    Space for Extended Chars is not used, nor Future Color
- User  gets 4k space - (but only 1K to be used*) (915 bytes)
    4K dynamic RAM would be cost the same.
```

Again: Was this an older design?

Work Block:
(See also /Introduction/Description/Notes)

```
  Address BITs      4k             2K             Options
/-------------\   Block          Section         /-------\
A15 A14 A13 A12          A11                A10
*------+---+---+ ------- *===*  ============ *===*  =========    - - - - -
:      :   : 0 :  ROM    :   :      Code     :   :
:      : 0 : - :  -----  : x :  --    +   -- : x :                R  (8K)
:      :   : 1 :  ROM    :   :     Tables    :   :
:      +---+---+ ------- *===*  ============ *===*  ==========   ---------
:      |   |   |         |   |   Chars       | 0 |-> (Colors)
:      |   |   |         | 0 |-> to    /---- |---|    ------      r+w (2K)
:      |   |   |         |   |   Screen      | 1 |->  Chars#
:      |   | 0 |  Video  +---+  ------------ +---+  ----------   ---------
:      |   |   |         |   |   Pixels      | 0 |-> (#80-#FF)
:      |   |   |         | 1 |-> for   /---- |---|    -------     W  (2K)
:      |   |   |         |   |   Chars       | 1 |-> #00-#7F
: 0 0  | 1 |---+ ------- *===*  ============ *===*  ==========   ---------
:      |   |   |         |   |       / echo  | 0 |
:      |   |   |         | 0 |        ------ |---|    (4K)
:      |   |   |         |   |       / echo  | 1 |    USER
:      |   | 1 |  Data*  +---+        ------ |---|    AREA        r+w (4K)
:      |   |   |         |   |       / echo  | 0 |
:      |   |   |         | 1 |        ------ |---|->              - - - - -
:      |   |   |         |   |   1k  / echo  | 1 |  1k present   StartAddr
:      +---+---+ ------- *===*  ------------- *===*  ----------
       : ^   ^            ^                   ^
       : 8k  4k           2k                  1k
       :
```

The above makes the first four 4K blocks (lower 16k)
Note: See Roads not Taken ... But before you do:

```
!!! On a plain unit, 1K for user means 900 bytes to work with!
    Plus a minimum of 30 bytes for both stacks, being considered.
    The ACE hardly crashes, but near 900 Bytes will refuse to Edit.
    Can learn, can experiment, FORTH is very economic. Little more.
```

So yes: 4K was needed (even a 2K option would avoid much frustrations).
The ACE 'failed' not for any of the ‘reasons’ argued. Those were attractions,
Nor by flaws introduced (bumpy keys). It failed due the 1K limitation imposed.
This 'feature' has travelled faster than “FORTH” is.

And 25 years later (on Wikipedia, that incoherent thing) it was still being diminuished:
- "It's 3x faster"... a controller insisted, erasing the 12x reference (now proven better).

Video Workings

How is Video Data managed?
VIDEO represents most of the Jupiter's Hardware. Circuit is fairly simple, beyond the apparent confusion schematics impose. Clear or a mess depend on how these are presented. The VSynch calculation circuit may be hard to follow (on a first view or without training).

Video controls access to its own 2K RAM, consists of 1k Screen Buffer and 1k Graphs Buffer. The 'trick' being to allow safe access by the CPU to the Char Definitions Buffer (1K=128 chars) and user access to the Screen Buffer of ScrChars=1K=ScrChars.(could be ScrChars+ScrColors=2K)

Colour is misunderstood, and mystifyied. Crude Colours are not TV. It's so simple we wonder why High quality TV is usually argued: Colour is a question of phase, expressed as a delay. A CharColour is the corresponding delay applied on the signal sent... Just that!
Yes, Colour doubles a small video circuitry. Reason for video to be modular in order to allow a replacement.of BW to colour. Savings come from 4K RAM, half price of 1K+1K.

Priority Signal (Area)
Each byte can be accessed by 2 different addresses. For this, bit A10 is used as a priority flag. This scheme is neither really useful, nor needed. Video Hardware should had the priority. Always! So to avoid Video flicker when DMA capabilities are not present. Opposed to this, Tape routines need to be taken in account as they use this space with Priority over Video (not a real problem). Eliminating this signal would allow 256 chars (also allow a colour attribute (4+4) to every Char.

Screen Array (Image)
The 768 bytes of video memory is accessed by the ROM using addresses $2400 - $26FF.
This gives priority to the video circuitry, needing this information to build the TV picture.
The byte at $2700 is set to zero, to ease detection of an end-of-input condition.

The 254 bytes remaining are used as PAD (a Forth workspace), placed there for survival reasons. This same area is used by the tape recorder routines to assemble tape header information. It is not an absolute necessity of Tape code to need priority over the video circuitry (for accurate tape timing). As is, priority addresses $2301..$23FF are used.

Note: A private Video fast access is not imperative IF Video contention could be suspended using a small latch instead, as used to change addresses (as CP/M does). These latches are known as simple and cheap method for total control of a Z80.
((Away from the limitation of a ROM on the bottom, forcing fixed RSTs code.
((A no-no from the ZX-80, kept on a Spectrum corrected the imposed Video CPU waste.
((Kept here too, regardless the opportunity.

CharSet (8x8 Tiles)
Similarly, the Character Set is written on the 1K section addressed as $2C00..$2fff.
The video circuitry constantly accesses this, using 'echo' $2800..$2BFF to build the TV picture. Due to the dedicated nature of this space (ie, its constant usage by the Video Hardware) it is not possible (by design) to read BACK the data posted there (nor needed). Only to write it.
Mosaic chars where available for plotting in 4x4 'dot' squares. This by using a total of 16 combinations of the mosaic (8 mosaic chars and their 8 video inversions).

A note on Sound (software driven)
In the absence of a sound chip (as sound is harder than video) audio capabilities were restricted to programmable frequency and duration beeps sent to a small built-in speaker.
Write to any I/O even addressed Port pushes the diaphragm until any I/O even Port is Read.
Written Bit 3 goes to tape (thus a silent speaker, always ON regardless of value).

There's hardly more to say, except the software driver becomes harder to build.
(Maybe that ingenuity can do much with such limited, software driven devices.)

User Space

HOW does ACE FORTH uses the available RAM?

Note: 1K in Forth is equivalent to near 3 times more in a comparable BASIC language system).
Since the higher 4K RAM address bits A10 & A11 are absent in 1k RAM, the 1K RAM chip echoes
into the whole 4K space, as would on a breadboard (a suicidal circuit design).
So, it should be removed to allow usage of this 4k space by an upgrade of the ACE.

User Area starts at $3000 (12K) addressed to $3C00 (15K) to allow continuity.of 1K
a) The first 63 bytes is the System Variables Area ($3C00..$3C40)
These hold information as the number BASE and CONTEXT, the plotting coordinates
should the user wish to develop a word like DRAW to draw lines, and more.

Followed by User Dictionary:
b) This starting with the special "FORTH" vocabulary word. It makes connection to
words in ROM. As 1st Vocabulary it allows to reach FORTH User new entries.
...
c) Followed with User new Defined Words, if any already.
...
d) Next comes a GAP of 12 bytes, sized 3 Doubles.
It's an empty space to facilitate new words creation and also
a safety frontier against Data Stack underflow (thus 3 doubles).

User Area ends dynamically sharing what remains (Both Stacks):
e) The Data Stack itself, growing upwards, software driven.
f) At the opposite end (of free memory) is the Return Stack, growing downwards.
Used for threading (starting with FORTH interface loop, aka outer loop).
Note: This loop waits for commands to run, before and after a command.

This stack ‘sharing’ is unusual. Here needed, being more efficient than fixed partitions.
We know no other FORTH using this strategy. It reflects great care and ingenuity, also
allowing the user to enjoy the full 900 Bytes instead much less (as 1K replaced 4K).

Note: SysVars, 1st Voc, GAP and Stacks demand near 100 bytes
Fixed stacks would mean some 200 bytes, not allowing heavy recursion programming.
All free space is BOTH stacks, not limited by partitioning (allow free recursion).

===

Trivia: With 1K, user space is reduced by the above mentioned. It's ~ 940 bytes)

Note: Even with FORTH allowing 2x to 4x more code, with 10x to 20x the speed.
Still, 4k was NEEDED. 1k static RAM available was a stopper, that having
the same price of 4k dynamic RAM.

Did the circuit implemented the testing breadboard? Was dynamic RAM tested?
Need to ask, as the circuit (slowly hand-made) shows absolutely no options,
as if dynamic RAM was not considered. Due the importance and low price,
not being available imposed the use of a much more expensive expansion
by design, against the Jupiter ACE goals. (...)

===

Trivia: A TV screen of 32x24 chars occupies 768 bytes (and User RAM available is ~900 bytes).
Comparing, each page of this text has an average of 80charsx50lines, equivalent to
~5 screens... Each page of this text is weighting an average of ~4k (!)
Forth inside (Z80 ASM), is ~5K (including 1K headers). Thus, 4K too.
Wow!

Things not done

Long time ago...
On those days, development to Z80 was done on CP/M machines. Surely the case of ACE-Forth.
As such, ROM would be expected at the end of RAM-space. That would ease code move and its ‘burn’.
It would make sense to avoid the ZX-80/81 architecture, not to use its "fixed device" limitation.

That previous architecture was a nuisance partially corrected by using dedicated Video.
Surely proposed to the ZX-80, ‘strangely’ replaced by a damaging and hard to implement,
hybrid software/hardware solution later abandoned (not fully corrected on the Spectrum).

The other error was ROM location, as the Z80 followed the 8080 'vectors' position.
As a general computer it would make sense to place ROM far away. On end-of-space.
It would made sense to have Video there too, as well as the Write-only Char-Set.

So, it Would make more sense to build RAM on Bottom (more able). That would not cost more
thana few pennies more... Relative position of CharSet, being first, MAY indicate a try.
What was built is more than "pretty straightforward" (confession made because not own?)

It was anachronic, as mimic of a ZX-80 proposal that never was. We see no work there.
We see development stopped, replaced. As Video replaced then expensive Video-Chip.
(The Commodore Video-Chip was to replace a bigger circuit. Sold, was expensive.)

Reasons, reasons...
The soul, of the little beast, that is another thing. Its code shows an evolution.
It shows hard work, even if over ideas present at the time. Some hard to implement.

Due the fast pace of FORTH evolution, more recent Z80 solutions moved away from
the Z80 very limited 8080 heritage. Because recent, the ACE started with the older
FORTH implementation (maybe FIG 1.0, as FORTH-79 implementations were harder to get).
Why did it was not rewritten? Why was it kept? ROM code shows how that path made difficult
the innovations written later. By then (only then) the start made the rookie a Forth wizard.

<u>In short</u>, the FORTH STACK should have been translated to the CPU stack. Period.
Needed changes to allow automatic disassembly become very hard, also reason for
(those particular areas of the ACE code) to be so hard to follow. Thus hard to develop,
hard to implement. And we can guess a true nightmare to debug (we do testify that much).

<u>Consequences</u>: Hard development, sure delays by using a Soft-Stack as Arguments-Stack.
Reducing speed was from the ~33xBasic (Sieve=47x) of FIG 1.1, down to ~20.8x (Sieve=20.7x) as our
measures show and our private (wider space) optimised ROM patch proves, optimising essential OPs to
deal with the soft-stack. Surely that was done. then lost. Lack of space (5K) forcing not to use these
Worse, 5K forced to reduce vocabulary and to reduce some primaries into secondaries, reason for the
final ROM speed down to ~15.7x (Sieve=12.4x)... ! See "The ACE ROM, patches and Changes" section.

Note1: Final ROM speed values can change with loading restored words.
Note2: Measures made with BYTE primes Bench become inaccurate, indicating 11.6x
(or 12.4x, FILL word dependent.) None of these values are correct, as later found.

A short comment on speed:
We took time to build an more accurate measure (this is possible with FORTH) using average* usage
of each FORTH word, by weights. (*Straight Usage Average, or frequency. Not squares of usage.)

'Weights' measure is much, much more representative, accurate, and stable.
It avoiding the ROT word. Also FILL word is removed with 2 results:
- Fair to Basic (not having FILL), where Forth would benefit.
- Much more stable conversion, due the reason above.

‘Weights’ measures, either raw or converted to "xBasic", depend from a Words Average Usage list.
There are papers available on the mater of frequencies, last we used are from the authors of GForth.

An 'amateur' solution

Choices needed, for flexibility and quick development: One *week work (ASCII schematic cleaned)*

```
$0000#=========0=============================================#      #============#
     + $0000   |  RSTs, Vsynch, Check and Debug Vectors      |      |            |
     + $0100 --|  User Area (SysVars) = 256Bytes             |  4k  |   BASE 0   |  Base version
     + $0200 --|  User Dict (+stacks) = comfortable 3.5K     |      |            |
$1000#=========1=============================================#      #============# -------------
     :         |          + 4K RAM ready Socket              | (4k) | Expansion1 |  Plus version
$2000#=========2=============================================#      #------------#
     :         |      + 4K RAM circuit to add Socket         | (4K) | Expansion2 |
$3000#         3    ( Plus version should have sockets )     |      |       -----|
     :         |      + 4K RAM circuit to add Socket         | (4K) | Expansion3 |
$4000#=========4=============================================#      #============#
               :                                             :      :            :
               :              +16K Bus RamPack               :      :            :
               ---RamPack may need to slide down Addrs 8K ---  32K  :   (open)   :
               :              (+32K Bus RamPack)             :      :            :
               :                                             :      :            :
#====#=========C=============================================#      #======#=====# -------------
$Cxxx:-        |    4K free (Socket or Pack) for ROM or IO   | (4K) | read |write|   12K VideoX
#====#=========D=============================================#      #======#=====# ----- option
$Dxxx:-        |    4K ROM-R ACE FORTH1 || Xtra-VIDEO  (W)   |      | ROM1 |write|      \
#====#=========E=============================================#  8K  #======#=====#       \  (W)
$Exxx:         |    4K ROM-R ACE FORTH2 || Xtra-VIDEO  (W)   |      | ROM2 |write|  (R)   \
$====#=========F=============================================#  +   #======#=====#         -----
$F0xx: select  |  3K SYS sys routines   || Tiles1 on W  1k   |      | BIOS |Video|  12K Firmware
$F4xx: R || W  |              on Read   || Tiles2 on W  1k   |      | read |write   (work:11K)
$F800:_________| ________________"_____ ||-Scr2   on W  1K-  |  4K  |___"__|- " -|       +
$FCxx: latched |  boot(reloc)+init>>off  == Scr1   on RW 1K  |      |      |  RW |  (boot: 1K)
#====#=======================================================#      #======#=====# -------------
```

Note the relation (the direct exchange of code, both ways) with CP/M + Access control,
for boot swaps and ROM/Video access depend only on a pair of cheap-chips... Pennies.
Isn't that what engineering is all about?!? IE: making the most, with the least?

A single 4K chip was enough for 256 tiles plus 2 Video pages.
Since dots are sequenced in packs of 8 bits, Video subsystem would have enough time to alternate consultations of both SCREEN and TILES. Naturally, access by the Z80 would still need to be buffered.
This removes the cost of 1 dynamic RAM, and adds two extra (cheaper buffers)... All in good balance!

An improvement ON THIS would be to use the 2nd page for colour, directing the delay needed by the colour phase... Interesting subject, the TV colour protocol: After uncovering all the ‘theory’ thrown over, mostly misdirecting. Why to do so? The above scheme allows expansion to colour!
(And yet, still much less work than the one ACE-Forth evidences. Check Listing on Book2.)

Maybe because, what is simple, is not valued. Nor hard work to achieve it. Nor even to protect it.
Very few people respect the heavy work simplicity demands. Most do not see it, just say (done with too much confidence): “I know how it works” (then confusing ‘description’ with what stays behind it).

All becoming 'acceptable'. Even when for mutual delusion, specially on other areas.
'Sure' delusions (supposed competent) can be extremely dangerous, as in medicine.
There, delusions (under and for a pose) can be deadly. Can inflict much pain.
Delusion also serving as weapon of mass destruction ('sure' of the inverse).

___
 # "The highest form of human intelligence is to observe yourself without judgement."
~ Krishnamurti

-

1.3 ACE Firmware

- ► Enter Forth
- ► Missing, yet available
- ► *Roads not taken*

"The Ghost in the Machine"
Forth is BOTH an OS (an interface) and a Compiler, kind of ASM meets Pascal.

ACE FORTH is a FORTH-79 sub-set implementation, with extensions. Apparently started with FIG 1.0 8080 Docs.
((This version no longer available, we suppose this because 8080/Z80 programming wisdom directs its kernel.
((Also because F-79 is a set of directives, while FIG presents code choices found on ROM and on the Manual.

As FORTH advantages started to be known, ACE's innovations come as a bonus:

- FORTH was **fast**, above 15x to 33x faster than the BASIC most used at Home.
  (!) Not exactly the Byte Bench measure of 12.4x. (ACE Architecture can go 22x)

- It allowed **easy ASM** programming when needed.
  (!) ASM being the overlooked Left-Hand of FORTH.

- Enforced **structured programming**, not obscure spaghetti code.
  (!) A better description would be an interactive structured assembler.

- Not the least, ACE-Forth allowed **edit/redefining** of words without using sources.
  (!) FORTH is disk-oriented System/language. No disk would demand a RAMdisk, wasting RAM.

A move into Non-Disk demanded more:

- Disassembly of User Words (program pieces) to allow the cheaper Tape-Recorder solution.
  It filled the absence of usual to Forth, but expensive, Disk/Tape drivers.
  (A cassette-tape is NOT a tape-driver. It's not useful for repeated access.)

- Simplified usage, allowing to edit high-level type definitions (as ARRAY).
  Allow adding structural compiler words (as CASE-OF), easing to the user chances
  usually reserved to gurus. BTW: This is a chance also related with disassembling.

- Integrated Floating Point operators, not needing to buy a library when needed.
  Not much used, if at all. But a general request, that boost was felt needed.
  Maybe a bad move, maybe not: Loaded would fill available RAM. Had to fit!
  Fair Notes :
    SPEED: Against FLOATS, FixedPoint math is much, much much faster on 8bit CPUs.
    PRECISION: FLOATS were expected. Simpler, they expand and replace DOUBLES math.

ACE-FORTH was reduced to be a 3K FORTH to fit 5K available, barely offering a chance to evolve
into a more common the 6K FORTH (plus 2K of disassembly/usability extensions). Full 6K Forth can
be achieved with a RamPack (or an emulator). Cheap ROM should had been 12K. Cheap RAM: 4K dynamic.
So, why to place the missing words on expensive RAM, instead on cheap ROM?. Is this not a nonsense?

For the (ROM unaware) User, the absurd was the 1K static RAM limitation. It was painful.
Not at start, when learning Forth. For advanced usage, it meat ~900 bytes available (!)
This was THE single major reason (overlooked) for the Jupiter+ACE commercial failure.
Other reasons existed, secondary, as keyboard.built as uncomfortable (and failing).
The absence of colour is mentioned, not a reason. Optionality would cost pennies.

ACE-Forth was a feat... Was a promise delivered.
We still remember that beast for its firmware... We do honour it!
Its hardware was another matter, as the reverse of the Jupiter ACE coin.

Enter Forth

WHAT is FORTH?
It's a compiling system that runs what it has compiled. Also growing with it. Period!
Simply put, it was the only compiler (not a batch) compact enough to fit on so little.
Structured, it also allows easy build of ASM routines (a welcome feature to 8bits).

Why so different from Pascal or C? Yes and No...
<u>Yes</u>: It's interactive, can act as an OS. (It also promotes system <u>awareness</u>, not hiding it)
<u>No</u>: As language it just not hide stack usage from the programmer, just as Assembler does not.
<u>Why</u>?: Compiler simplicity, efficiency... Such hiding is an heavy task to every 'pure' language.
Again yes, is more like a Structured Assembler than C (an old C joke). <u>Argument passing IS manual</u>.

Arguments are a Programmer's task! (as in ASM).
Forth uses a Data-Stack in place of CPU registers. It's for function Arguments, temporary values.
Compilation is simplified while being a Structured Language (easier than Assembler).
It's a Threading immediate-pass Compiler (of addresses or subroutines).
Threading reduces Code size while also avoiding interpretation management.
Using a Software Dispatcher slows things a bit. More on the Z80, a CPU not particularly fit for the job. But it does allow a great versatility for a small cost (still fast on the Z80).

Advantages of Programming with an Argument's Stack
* Code is easily re-entrant. (No data interferences.)
* No Argument management interfering on procedures (Snippet code autonomy.)

Both keep compilation simple, efficient and structured (Effective on cheap CPU's)

Args Stack Disadvantages on Programming:
* Many ways to do the same (a confusing advantage).
* Mostly of management (with the advantage of efficiency).

Some practice is needed, knowing what's what. (Also builds ASM skills, used or not)
NOTE: The stack is NOT a replacement for variables. That's a common erroneous approach suggested by a double personality of the Argument stack: It also serves procedures with temporary values not demanding an explicit local variable (still possible).

It's a good practice <u>to reference stack state</u> when developing a new 'Word'! To see the args while these change or float around. Should be referenced when that 'floating' makes them not so obvious. To keep all manageable, a max of 3 stack values is considered. If needing more, local vars should be used as any other (language) compiler would: Carefully pushing them to Return-Stack, cleaning the whole before Exit. On the ACE those locals are invoked with I, I' and J.

More interesting characteristics:
* Data structures are easily built (as Pascal types or C structs)
* Command set is expandable, either as user words (procedures) or as primary words (ASM).

We need to stress this: Forth's twin brother is Assembler. The CODE word is Forth ASM link.

<u>A bit of unwritten History</u>: FORTH began as microcode. *(Followed by an auxiliary reader/runner.)*
Then slowly growing from the lessons taken, on idealizing a CPU code-set, into an universal tool.
(Chuck Moore interviews imply that much. We can visualise its growth as an evolving Assembler.)
A 2nd Stack to pass data arguments, gave the freedom allowing the OP-set to be refined.
A crucial element was moore needed: A Chuck More maestro (as toddlers need much care).

LISP was an influence, not a driver. Growing independently, by goals, Forth was moore practical.
Many similarities (not stated either) show the <u>influence</u> of LISP on its evolution. The 2nd stack was a simplification due its practical nature: Forth can be seen as using limited 'lists'.
'Words' were simplified Lists made after a single Node, the CFA. Refinement continued, free as the Sky, until valuable as a Diamond (called FORTH, independent, not uLISP).

___
 # As the DNA structure was grokked, so was Forth! ... A smaller brother (sons of Lucy in the Sky)
FORTH was a Beatle language, not pretending to fly... Yet Flying. Not crawling from other chairs.

A language overlook

As suggested, Forth promotes programming awareness, exercise of mind and sight..

We may guess corporative tycoons may hate that, rather keep dependencies and obscurity.
Efficient as it is, Forth tends to be dense in the sense of far fetching, with many actions available or possible instead 'legal' constructions. We see it as a complete, universal tool.

ACE-Forth novelty was to be well adapted to Computers without Disks, or Tape drivers.
That made it unique, also making available a real language (even if reduced).to the public.
While the full Forth-79 Word-Set is not present, the crucial words are. As bones allowing to add any muscles missing, as to build later Forth-83 words. The essential, and management, are there:

* **Data Structures:**
Integer, Floating point and String data may be held as:
  - constants,
  - variables
  - arrays with multiple dimensions
  - mixed data types .

  and easy creation of new structures defining words (with DEFINER)

* **Control Structures**, all 'nestable' to any depth. :
  IF-THEN-ELSE,
  DO-LOOP,
  BEGIN-WHILE-REPEAT,
  BEGIN UNTIL,

  and easy creation of new control directives (with COMPILER)

* **Operators:** Arithmetical, Boolean, Comparison
  all extendable or added by the programmer (or user).

* **Program Editing**, made practical without disk (built as a coherent solution)
  - FORTH words may be listed, edited and redefined. (An ACE-Forth great characteristic)
  - Comments are preserved when words are compiled. (With little or no impact on speed)
  - Internal to the system LIST, EDIT and REDEFINE (Are an integral part of ACE-Forth)

  So: the system is a development able system even if not(1) complete. A Forth base-system relies on the 'Forget' word to be independent of an edit program. The available 'Edit' fills the user needs of a change, complemented with 'Redefine' (avoid forget&retype).

* **TAPE storage Management.**
  - User Words an Vocabularies.
  - Binary Data (RAM Space)
  - Source by disassembling.

  To use common tape recorders, instead disk/tape drivers

<u>Note 1</u>: Forth reduced to a ~3K Forth with ~2K needed extensions:
Extensions: Disassembling+Edit, and Floats
Remaining 3K replacing a missing system-ROM,
System: Console, CharInit, Sound, Tape.
<u>Note 2</u>: Options do exist, to extend ROM, even to patch ROM.
However, that supposes a ROM burner. The patch option demands getting some extra new space, either by reducing Char Tables, either by removing the FLOATs Library (it could be loaded).
<u>Note 3</u>: FLOATs were a welcome addition because demanded. BUT did become a weight to ACE FORTH, responsible for the loss of Vocabulary Paths (defined but removed).
And for the extreme reduction of some needed Primaries, then turned Secondaries.
<u>Note 4</u>: Choices without time, can be bad choices.
(As most administrative choices are)
<u>Note 5</u>: Still a joy!

Word List

A first full view of available words can be, IS intimidating.
And yet, each word exists only to serve a purpose, when the need exists.
Relax! Read the manual. Walk with it, visit the different tools available.
A new territory is to be visited. Known slowly (tourist today, guide tomorrow).

VLIST content, grouped by activity (immediate words are underscored)

FORTH	BASE						
VOCABULARY	CURRENT	CONTEXT	DEFINITIONS				
VLIST	FIND	LIST	EDIT	REDEFINE	FORGET		
HERE	ALLOT	C,	,	LITERAL	ASCII		
DEFINER	COMPILER	DOES>	RUNS>	CONSTANT	VARIABLE		
CREATE	:	;	IMMEDIATE	[	]		
!	@	C!	C@				
IF	ELSE	THEN	LEAVE	I	I'	J	
+LOOP	LOOP	DO	EXIT	BEGIN	WHILE	REPEAT	UNTIL
ABORT	QUIT	EXECUTE	CALL	(			
DROP	DUP	SWAP	ROLL	PICK	OVER	ROT	?DUP
R>	>R						
+	D+	-	ABS	MIN	MAX		
1+	1-	2+	2-	NEGATE	DNEGATE		
U/MOD	U*	*/	*	*/MOD	/MOD	MOD	/
0=	0>	0<	=	>	<	U<	D<
XOR	AND	OR					
RETYPE	QUERY	LINE	PAD	TYPE			
HOLD	#	#S	SIGN	#>	<#		
OUT	IN	INKEY	BEEP	CLS	PLOT	AT	
CONVERT	NUMBER	QUERY	WORD				
."	EMIT	CR	SPACE	SPACES			
DECIMAL	.	U.	F.				
UFLOAT	INT	FNEGATE	F/	F*	F+	F-	
LOAD	SAVE	BLOAD	BSAVE	VERIFY	BVERIFY		
SLOW	FAST	INVIS	VIS				

F79 Words missing (most already on our library)

+!	CMOVE	FILL	COUNT	-TRAILING	
EXPECT	>IN	KEY	NOT	STATE	DEPTH

Primary Words downgraded to Secondaries (needing restoration)

OVER ROT - < = ABS MIN MAX

TIP: Most of the downgraded words were restored, present on the Primaries Library
The most essential are already restored on the patched Roms.
Use the patched Roms. Either on an emulator, or burned as an upgrade.

Fair Warnings:

F79's FIND is now F83/ANSI ' (tick) ... ASCII is now called **[CHAR]**
... NEGATE is neither the bitwise CPU Op 'NOT', neither a Boolean 'NOT'
('NOT' is a funny problem, teaching us to look instead to listen)

Missing, yet available

The Hardest Question...
- What's worse? To loose words from a full set, or to un-optimise essential words?
We see both damages on the code. The ACE deserved to be a CP/M program, not because of 8K.
But because that 8k had to include system, to be just 5K (including extensions).... And yet ().

It's not a matter of previsions that fail. That may fail, even on extra care, due hidden pressures.
It's about not to predict safely. It's about "something else". And yet, all the essestial is there.
Able to restore the missing what's missing, the essence is saved. Nearly forgotten, it's to remind.

We loved the ACE in spite being limited (not a 1st consideration), we regreted the Jupiter.
*Not (just) because keys failing, but first and mainly because its 1K instead 4K. Even so, due both, I had my first job after showing in a word how keyboard worked. Better: Having experimented the essence of programming. Saved millions to my country, in a project I was integrated **---** No, was never rewarded. More would happen after that (the usual opposite).*

The ACE did its job, a didactic tool to structured programming, very well.
Speed was ok. With it we learned to keep it so. What was learned become a second nature.

Could ACE-Forth be better? Not an issue when we can replace most words. (Here we show that much.)
A few essential words are available on the next pages, more on Chapt 4.4 "A Primaries Library".

After all these years, we examined ii critically to know, to see clearly. We found clouds too many.
We share here what's available for you to exercising awareness (if the invitation is accepted). Never give 'opinions' (social impact delusions): "Rather look, then to see!", is our practice. Then suggest you to take the bull by the horns, gently. (You are welcome, send us a postcard).

Some Missing Words (standard)
+! CMOVE CMOVE> MOVE FILL R@ CODE NOT
' EXPECT COUNT **-**TRAILING >IN KEY DEPTH
STATE [COMPILE] COMPILE
The Cripled Words (turned Secondaries)
As important as words missing, is being crippled to fit in.
OVER ROT ABS = < **-** MAX MIN
And the Other (defacto used)
Some important Primary Words (also in the Missing Primitives Library)
2/ 2* @EXECUTE LSHIFT RSHIFT NIP TUCK
Useful Words (occasionaly used) (available in the Full Primaries Library)
RANDOM 0 1 -1 2

The ACE survived many undue cuts. *Barely.survived strict minimum. It was so close not to fit on 5K, nor to work on 1k. It may have been considered to abort the project. A close call on distorted conditions, and Floats support (650 Bytes) in place of Doubles (no longer both). It survived the underdeveloped hardware where we can see an old solution (as the schematics and the Memory Map testify). Lost the success it could had been, its Forth is still remembered*!

On the useful Words removed
Curiously, zero was present (still is) on the ACE code. It's important for shorter and faster code, thus present on ROM. The manual mention several words that would be present IF there was space.

Traditional RND routine was presented in the manual. By the 90's, a new RANDOM method was available.
We've implemented it, it's on our GPL library. (It's unique, demanded hard work to make it right.)
Try it! Do something! ... Enjoy its speed, sometimes needed. *It was done for you. (Postcard ?!?)*

Our method to get full the benefits of VOCs :
```
0 COMPILER [FORTH]  FORTH  RUNS>  ;              \ On Edit-> context. On Run -> ignore
          vocabulary :UTILS:                      \ Name Vocs inside colons. Then their
          0 COMPILER [UTILS]  FORTH :UTILS:  RUNS>  ;    \ compiled invokers  inside brackets
```

A few missing Secondaries

```
 : \        QUERY  CR  ;  immediate            \ standard comment to EOL... Not compiled
 CREATE \\  HERE 2- FIND \ swap ! immediate     \\ non-standard, more visible than a single slash
 : U?       @  U.                               \\ DEBUG: Examine a Var's content as Unsigned (ptr)
 : T?       COUNT TYPE ;                        \\ DEBUG: Examine a TextVar (tArray)... Ex: PAD TEXT?

\\ Important FORTH WORDS  (As defined in the next section. Thus commented here)
\\  : HEX  [ DECIMAL ] 16 base c! ;         \\ Standard. Needed to avoid errors.
\\  : CODE  create here  dup 2- ! ;         \\ Build a primitive word
\\  : END-CODE  [ HEX ] e9fd , DECIMAL ;    \\ compile NEXT to end a CODE word (we will not use it)
\\  : R@  I' ;      \\ As Secondary, get 2nd item (it's call). Wait! Better as a Primary (later)
\\  : 0   0  ;      \\ Shorter zero call    Wait! Better as a Primary too (later)

\\ As string words had to be removed. Say this because they are standard (did not fit)
\\ The ACE manual suggests to define some of the missing, as an exercise. Have them here.
\\ Anshor explanation:Strings are Counted Arrays, Strings are work managing pairs.
\\ Inline strings,  as ."   are \\ implemented for the ACE (fully editable) with our  S"
\\ (Check "Managing Strings" under "Programming tips" )

 : /STRING  ( pos, size, adjust -- pos+adjust, size-adjust )    \\ Adjust string (ptrs, not content).
    >R  I -  swap R>  +  swap ;

 : -TRAILING  ( addr length+ -- addr length- )  \\ Eat-End-Spaces  (Pointer, not content)
    DUP 0       \ adr len  len 0
    DO                             \\ from zero to len... What if len is zero?
      2DUP + 1-                    \\ "2DUP" (2 args copy) is faster than the "OVER OVER" hack
      c@ 32 -                      \\ Quick test for space (ascii 32).
      IF LEAVE ELSE 1- THEN        \\ The Trick: A zero IS NOT a space, loop will leave
    LOOP ;                         \\ Apparently simple, a tricky excellent exercise.

\\ Standard on F83
 : NOT      0=    ;                \\ Boolean:  0(False)=>True(NZ)  and  True(NZ)=> False(Z)
 : ?        @  .  ;                \\ Content:  Common Var's content consultation (integer)
 : WITHIN   ( n1|u1  n2|u2  n3|u3  -- flag )        \ A solution for BOTH Ints and Unsigned
    OVER -      \\ n1  n2  n3-n2     |R|            \ n3-n2 is flat range (an abs value)
    >R   -      \\ n1-n2             |R| n3-n2      \ n1-n2 is flat distance (abs value)
    R> U< ;     \\ flag(n2<=n1<n3) , ie: distance Bellow the range = fitting within range

\\ *** Adapted/added to the ACE system
HEX 3C3B CONSTANT SPARE  DECIMAL
 : SPARE    [ HEX ]  3c3b  @  ;     : SP_   SP0  SPARE ! ;  \\ A SysVar.; SP_ resets Stack

\Now we can define the most common (and standard) debug Forth words mentioned in the manual:
 : SP       SPARE  @  ;            \\ First empty slot address above Stack
 : SP0      HERE  12  +  ;         \\ Stack address, empty or not (Beware, it may change)
 : DEPTH    SP  SP0   -  2 / ;     \\ 2/ found in the next pages should be used instead
 : .S       DEPTH  0>
            IF  SP  SP0
                DO  I  @  .  2  +LOOP
            ELSE  ClrStk  ." stack empty "
            THEN  ;

\\ Useful ACE system Words
 : KFlush   begin  inkey  0=  until ;     \\ Clean unattended Keys. To use before a Kwait or a KEY.
 : KWait    begin  inkey  until ;         \\ Wait for a Key. No ASCCI fetched, just proceed.
 : KEY      BEGIN  INKEY  ?DUP  UNTIL ;   \\ Wait for a Key. Stack ASCII. Continue.
 : spKey?   Inkey 32 = IF 1 ELSE 0 then ; \\ Do not wait, just flag (Space is Break, not an Abort).
```

Some missing Primitives

```
: \ query ; IMMEDIATE
\ Essential Lib is distributed under GPL3 licence, built by ©2017,2021 Dutra de Lacerda

\ GENERAL ASM UTILITIES
  : CODE create here  dup 2- !  ;                     \ The real deal, not an example, costs 0 Ts
  CREATE R@  find I  here 2-  !                       \ A faster and bodyless definition smaller)
  CODE HEX  16 base c!  103e ,  32 c, 3c3f , e9fd ,   \ HEX helps to avoid errors (here, smaller)
\ Following Words are useful, not critical:
  : [HEX] hex ; immediate                             \ Useful when already on compile mode
  : Next,  e9fd ,  ;           : End-Code Next, ;     \ Useful as formal definition of Regular-NEXT
  : Fnext, c3 c, 04b9 ,  ;     : FEnd-Code Fnext, ;   \ Useful for Words not altering stack size

\ PRIMARY WORDS MISSING from ROM

 HEX CODE  +!       \ 5.5x the Best Secondary  (Similar to "var++" on C, but with expressed inc value)
  d5df , e1df , 234e , 2b46 ,  09eb ,                 \ This Word should be on Ron
  73eb , 7223 , End-Code

 HEX CODE 2DUP     \ 12.3x (OVER OVER), 5.34x if OVER is our Primary.
  \ Mostly used as a 2 Arguments copy, while also being a Doubles DUP
  3b2a , 543c , 015d , fffc , 0109 ,
  0004 , b0ed , 22eb , 3c3b , End-Code

 HEX CODE FILL     \ 19.7x the Secondary
  7bdf , 42df , df4b , 5feb , b079 ,
  0528 , 2373 , 180b ,  f7 c, End-Code

 HEX CODE CMOVE    \ Warning: Direction may cause overlap. CMOVE> uses the reverse direction
  42df , df4b , dfd5 , d1eb , b178 ,
  0228 , b0ed , End-Code

 HEX CODE CMOVE>   \ Warning: As above... And vice-versa: CMOVE uses the reverse direction
  7adf , 20b3 , df04 , fddf , 1be9 ,
  4b42 , ebdf , e509 , ebdf , d109 ,
  ed03 ,  b8 c, End-Code

 HEX CODE  LSHIFT   \ value n -- value(n_shifted)
  43df , cbdf , cb23 , 1012 , d7fa , End-Code

 HEX CODE  RSHIFT   \ value n -- value(n_shifted)
  43df , cbdf , cb3a , 101b , d7fa , End-Code

 HEX CODE  2*       \ 3.2x the Best Secondary "dup +" aka 2* here replaced
  \ Use  n LSHIFT  not to repeat 2* n times... Also: Beware overflow to signal
  cbdf , cb23 , d712 , End-Code             \ Much used, should be on Rom

 HEX CODE  2/       \ 46x the Secondary (Signed values)
  \ Warning: Signed -> Arithmetic_Shift. Unsigned -> Use  1 RSHIFT  instead
  cbdf , cb2a , d71b , End-Code             \ Less used, to be loaded when needed

 HEX CODE  COUNT    \ 4.0x the best secondary (Actually a Multi_Purpose function)
  \ fetch and advance ( addr0 -- addr1 n) Mostly used for CountedCharArrays
  d5df , d713 , 5ee1 , 0016 ,  d7 c, End-Code

 HEX CODE  MS       \ Wait for MiliSeconds (Mono-Task version, 3.25MHz based)
 \  ANSI tool ( u -- )  Accurate MiliSecond delays from 1 ms up to 64 seconds
  06df , 10f8 , 1bfe , b27b , f720 , End-Code
```

Then, Restore Speed

```
\ This Essential Lib is under GPL3 licence... ©2017, Dutra de Lacerda

\ These PRIMARY WORDS were sacrificed. Restore them to Primaries

 HEX CODE OVER     \ 2.3x the Original
  d5df , 42df , d74b , d7d1 , 5950 ,
   d7 c, End-Code

 HEX CODE -        \ 2.06x the Original
  4bdf , df42 , b7eb , 42ed ,  d7 c,  End-Code

 HEX CODE ROT      \ 1.97x the original
  d5df , d5df , 42df , d14b , d1d7 ,
  50d7 , d759 , End-Code

 HEX CODE ROT-     \ 1.96x the Best Secondary
  4ecd , df08 , dfd5 , 50d5 , d759 ,
  d7d1 , d7d1 , End-Code

 HEX CODE ABS     \ 4.3x the original
  cbdf , 287a , af06 , 6f67 , 52ed ,
  d7eb , End-Code

 HEX CODE  <      \ 2.9x the original
 4ecd , df08 , afeb , 42ed , 177c ,
 579f , d75f , End-Code

 HEX CODE  =      \ 3.0x  the Original
  4ecd , df08 , ebaf , 42ed , 7c57 ,
  feb5 , 7a01 , 5f17 ,  d7 c,  End-Code

 HEX CODE  MAX    \ 10.4x the Original      \ Rarely used
  cddf , 084e , 21d5 , 8000 , eb19 ,        \ Should not be on Rom
  0021 , 0980 , edaf , d152 , 0238 ,        \ when (it was not) finished
  5059 ,  d7 c, End-Code

 HEX CODE  MIN    \ 8.9x Original           \ Rarely used
  cddf , 084e , 21d5 , 8000 , eb19 ,        \ Should not be on Rom
  0021 , 0980 , edaf , d152 , 0230 ,        \ when (it was not) finished
  5059 ,  d7 c, End-Code

\ Speed-up while reducing sources size      \ 1.3x faster, half size on source
 HEX CREATE 0   068a   here 2- !            \ Prioritary, this one still exists in ROM
 HEX CODE   1   0111 , d700 , End-Code      \ Not as critical as <zero>  still much used
\ Optional (commented)                      \ Words use 2 bytes, instead literals costing 4
\\ HEX CODE  2  0211 , d700 , End-Code
\\ HEX CODE -1  ff11 , d7ff , End-Code


\ Identifying in-range conditions should be fast. (A needed ANSI word)
\ It's an ANSI tool to simplify inclusion tests
 HEX CODE  WITHIN   \ ~7x best secondary    \ ( n  lo  above -- flag )
  d5df , 42df , e14b , afeb , 52ed ,
  dfd5 , 6960 , afeb , 42ed , afd1 ,
  52ed , 21c3 ,  0c c,

\ More complete set is available on "A Primaries Library" (Chapter 4.3)
```

Roads not taken

A glimpse on what 'did'
When the ACE was developed, Forth was known mostly from BYTE magazine.
Less known was how to work with it... Its insides, those were a mystery!
(Loeliger gave us a very good description. So did Dr.Ting, available later.)

ACE ROM code shows an evolution, from educated rookie into mastery. And yet,
that mastery was not expressed on a desired Stacks change, surely noticed.
Maybe because too much was already done, and the pressure of schedule.
Or maybe,also, because a quick change attempt failed. Then both.

That initial stack architecture is mostly due to 2 misconceptions very much alive:
• An obsession with the.(very important) Dispatcher, then overlooking the whole.
• Z80 best practice of keeping Registers available, unaware of Forth dynamics.

On Threading Indirectly, NEXT is of paramount importance.
But COLON/SEMIC (Enter/Exit) are not, because less frequent.
As important as NEXT, is the Data push/pop mechanism. Cpu-Stk pays!

The Z80 Stack costs 10 Ts(clocks) while the Soft-Stack costs 79 at best.
This 2nd stack MUST be used for less frequent usages, as Enter and Exit.
The balance of gains and losses on the Z80... results as a double speed.

What 'should', but did not.
Data stack should had been the CPU stack. Not the Soft stack previously used on the Z80.
The Return stack is less frequent, and Forth is not Assembler (Even >R and R> are occasional.)

That change was surely attempted. All code already done (the process is incremental) had to be fully redone. Everything. If our guess is correct, time constrains may (an educated guess) have forced the author to share the worry of loosing (maybe a month) rewriting code.from scratch.
Fact is no one can fairly decide except whom is inside. Outsiders ARE NOT ABLE to do so.

Surely (again) an unseen consequence (beyond the speed difference) was that without that change code would be slightly bigger (by a small margin). WORSE: Development would be harder.

Our estimation of time loss ON debugging DEFINER/COMPILER (a complex new system without parallel) may have been around 2 weeks of intense stress. (We've been there). Then hardened by a reduction of already small space available (due System/Console/Tape routines sharing FORTH space).
(This lack of space paradoxically justifies the FLOATs section, replacing DOUBLES)

Lack of space is quite visible, extreme, on turnning some crucial words into secondaries.
Example: OVER has 10 bytes as a secondary. As a Primary (code) it would occupy 13.
This battle for bytes also affected Primaries (no Soft-Stack optimization).
Example: SWAP has 10 bytes (code, not total). Twice fast has 20 bytes.

Trivia:
It's curious to see how the Char table was pushed to the end of ROM: A fire-and-forget technique.
With it, for the same reason, the RAM-ROM link, initially the copyright word (the first defined).
The usual print Company and author, still a link (a fair assumption). The 1 char name remained.

Top-of-ROM was the perfect anchor for hand adding on hard move (no CP/M-parallel) to ROM.
The anchor allowed not to alter too many pointers (and mistakes) until ROM was finished.
(been there). It ALSO means a few bytes missing, needed to avoid the last sacrifice:
... The OVER word (made secondary for 3 bytes, after ROT and 'minus')

A lot of time was needed, to check and recheck. To examine chances losses so the whole fit.

___
 # The ACE faced non-existence. Away from its goals it lived enough. Now dead, it still kicks.

Analysis

For all what is visible in the ROM code, its patent evolution, the chances left behind, supported by benchmarks anyone can verify, and by benchmarks we carefully executed or simply measured (again very carefully)... Reminding what was available by 1981, sometimes hardly...
For all the innovations present on ACE-Forth code...

... We can confidently consider the following :

In spite of all effort,
ACE-Forth suffered immensely from 'administrative' decisions and hardware absence of real development.

• First, by not losing a month on what we suppose to be half of its development. Why half?
  Its author accepted an administrative decision, on a competence only him had.
  We see, by the intricacies the disassembly process, it would save more time.
  We may suppose, a change was tried and failed. Thus the need to rewrite.
  Only this last point is an educated guess, based on the observed.

* Then, by not having a proper hardware support, limited to initial projections of size.
  Forth was known to fit on very small space (as a threading compiler). Full Forth, about 6K.
  BIOS was guessed to occupy 2K, including tape access. It occupied more (3K including Floats Lib)
  Innovations (disassemby and tools) would later need an excess of ~3K (not considering Floats.
  Unfortunately hardware was rigid, not considering expansion nor CP/M good example.

* At the end, his forced a cut on Forth available words, loss of time not used on the softer hardware.
  The 'cut' succeeded, though an unexpected delay. But incomplete, as evidenced by the presence
  of Max and Min words. And by the lack of optimizing the Push/Pull RSTs. Why?
  It's quite probable the trimming ACE-Forth delay. to fit the ROM, forced a delay on delivering it.
  That delay surely had a limit. Was the JupiterACE to be abandoned on the deadline?
  The trim process was/is incomplete, after a successful but still unclean ROM.
  Fact is, what was missed... would take just a few days, evidence of an hurry.

What happened? Why?
A ROM burn is NOT critical. Or was it? Fact is, a damaging schedule was imposed.
- That, would depend from administration an the hidden big investor.
  What we wonder, is why to keep incompletion. Why the refusal of a few days.
- Fact is, with the proper sources 1 to 3 days would be enough to un-trim the Rom.
  ... The un-trim would be more complete on 5 days (for a few more words, +! and 2).*

We took a bit longer, due the absence of master files and due a different Assembler.
- We had to adapt the listings (we used both the disassembly, and the restored listing).
  Fact is, adapting code to a different assembler is prone to errors. Even when changes are clear.

So we wonder... If the Jupiter ACE was expected to survive.
  Yes, it certainly is not, what it was supposed to have been in first place.
  It used a Soft stack, not being rewritten when due. That would be understandable.
  The ROM trim is incomplete, maybe as expression of (what's common). That's not acceptable!

Still today...
There are strange behaviours from ‘other’ anonymous administrators. Pretending to be a fan is a key.
Still blocking both history and facts. We noticed Facebook groups exist to ‘manage’ people, not facts.
  Not limited to the Jupiter ACE, general efforts to keep illusions and delusions as spread as possible, are worldwide. We do wonder if any sight is desired. As the installed void rules, we keep our sharing.

We all have a responsibility towards ConScience. Details being just details, we all should try to change attitudes without forcing, favouring sight instead word-of-mouth. An honest effort, shared becomes our's.

(On the next pages, the art of Benchmarks is used to evidence much. See also the external NG charts).

___
Fools rule the house... “Hell is empty, the daemons are all here” ~Unckown.

1.4 Benchmarks?

- ► ByteMag SIEVE (BASIC)
- ► ByteMag SIEVE (Forth)
- ► Workloads to WEIGHTS
- ► That Secondary OVER

Row, Row...
More important than speed (ACE FORTH become rather slow) are the innovations successfully implemented on this small introductory computer of the 80's (the “everyone can” class).
Still, speed was(is) important. A commercial print (quite good) stated in a small box:

FORTH Finishes First!
Speed Comparison Chart showing times in seconds to perform one thousand operations.

Type of operation	*Jupiter Ace*	*BBC Micro*	*Vic 20*	*Spectrum*	*ZX81*
Empty Loop	*0.12*	*0.67*	*1.3*	*4.2*	*17.7*
Print a number	*7.50*	*13.5*	*26.0*	*19.0*	*430*
Print a character	*0.62*	*1.3*	*3.1*	*7.5*	*24*
Add two numbers	*0.45*	*1.4*	*5.5*	*7.5*	*28*
Multiply two numbers	*0.90*	*1.6*	*6.5*	*7.5*	*32*

Because of the difficulty in devising exactly equivalent programs, these measurements should only be taken as a guide.

Fair enough! ACE FORTH was faster than the competition. But how much, that **was** hard to say.
Surely noticed: Redesigned (against Z80 practice), it would double its speed (33x BASIC).
... In spite its novelties, ACE-FORTH started with an old design. This was not replaced.

How fast is now known, better measurements have been made. 'WEIGHTS' is a FORTH-only Benchmark.
SIEVE Bench values were useful, to convert it into a (more accurate) Languages Comparison Tool.
Measurements with ‘WEIGHTS’ were translated using Sieve clean results. Then we got the answer:

ACE-Forth was **15.4x** faster (±1%) than the most referenced system... Sieve results were flawed!
Originally, we used ACE32 on DosBox, and EightyOne on WINE, with a small compound error of <5%.
Recently, we re-checked SIEVE values on a single emulator, reducing ratio compound error to =0%.
I.E. ***15.4x*** *common Basic,* **NOT the 12.4x** ***(rounded to 10x)*** *Sieve suggested.*
ByteMag SIEVE was precise, fairly evaluating 8 Bit CPU systems&languages.
Precise. Not accurate towards the final ACE version (due a small detail).

This **15.4x** is sadly far from the W=22x ACE structural limit. (Indirect-Forth limit is W=35x).
Got W=20x on a private patch by optimising critical OPs (just after restoring OVER, and ABS).
(These optimisations did not fit on 5K. Now they do: We rescued a few more bytes, to check!)

A FORTH Bench CAN BE more accurate than a general bench on other languages. Reason is simple:
Most words have a fixed timing. Thus can be seen as Benchmarks PER SE, after each % usage.
With their timing x usage frequency known... we can determine their compound weights!
Without further details, the result of **15.4x** become evident with our patched ROMs

What turned a precise SIEVE, so misleading?
ByteMag's Sieve Bench imposes the use of 2 less frequent Words (see End-of-this Sub-Chapt).
These were sacrificed to be Secondary Words (away from the whole), damaging Sieve accuracy.

After a final and fair conversion, ACE's global measure, shows to be quite higher than
what SIEVE 'suggested'. Let's not forget, as was, that Words can be redefined. Or loaded.
(Changing the Language is something FORTH does very well. Thus...)

Need a 16kPack? Best advice is: "Forget it" ... Use an emulator.
Better yet: Use our improved patched ROMs, and/or our restored Words.
ROM patching was possible by salvaging extra bytes from the BIOS Char-Def-Table.

ByteMag SIEVE (BASIC)

*** **ByteMag Sieve Benchmark** ***
Arrays on Byte bench were "0 based". ZX-Basic are “1 based arrays”.
Here adapted to “1 based arrays", carries the very same workload:

```
10  LETSIZE=8191
        20  DIM F(8191)
        30  PRINT "Only 1 iteraction"

        40  LET count=0
        50  FOR i=1 TO size
        60  LET f(i)=1
        70  NEXT i

        80  FOR i=1 to size
        90  IF f(i)=0 THEN GO TO 170
        100 Let prime=i+i+1
        110 LET k=i+prime

        120 IF k>size THEN GO TO 160
        130 LET f(k)=0
        140 let k=k+prime
        150 GO TO 120

        160 LET count=count+1
        170 NEXT i

        180 PRINT count, " primes"
        190 STOP
```

Hidden for decades, results have been a close guarded secret (we never saw the Sieve ‘adaptation’). Not “our thing”, Basic stated us cryptic errors. To outsiders these where a stopper. As found much later, Spect Arrays start on 1 (one-based)... As Sieve expected Arrays to start on 0, this also meant different adapted code: A naive adaptation would falsify Sieve results!

To be respect the Sieve, line 100 had to change to (i+i+1), no longer (2i+3).
Finally measured, and correct, the result was not surprising. But secret.
Initially extrapolated from 1 iteraction (only 1, to ease the pain):

10 iteractions = **79.6min** (4776 seconds, well measured)

Remind the ACE = **6.4 min** (a naive FILL would add 0.4 to this)
with OVER prim = 5.0 min (Forth Primaries are more natural)
and No Artifacts = **4.4 min** (simply put, the real ACE-FORTH)

These values were measured with a single emulator, allowing 100% accuracy of ratios.
(Previously, timings had a deviation of ~2%, compounding to ~5% ... Now replaced!)

Note1:
Since BASIC has no Fill, a fair comparison demands FILL to be removed from BOTH.
That removal also allows a better conversion of WEIGHTs, as confirmed later.
We'll examine this a bit closer, on the “Workloads and WEIGTHS” section.
Note2:
'Real' workloads changed a bit (3%), for having No-Artifacts (not just OVER-as Prim).
I.E., The conversion base (Weights<->Sieve) is now as close and perfect as it can be.

ByteMag SIEVE (Forth)

***** ACE needed words, recovered (faster)**

```
: \  QUERY  ; IMMEDIATE             \ Ignore Comments !!!
          : CODE  create  here dup 2- !  ;     \ Not needed on ACE+CODE.ROM
          decimal  16 base c!

          code FILL                         \ Restore this FORTH Word
            7bdf , 42df , df4b , 5feb , b079 ,
            0528 , 2373 , 180b ,  f7 c, e9fd ,

            : _FILL  ( Addr, N, Byte -- )      \ Example of a slower 'replacement'
              >R  OVER + SWAP
                DO  J  I  C!  LOOP
              R>  DROP  ;
```

***** ByteMag Sieve Benchmark *****

```
\ No adaptation needed, except to recover the FILL word, given above
\ Being Forth Word, FILL is a primary. For completeness, we also show a slow replacement.
DECIMAL 8190 CONSTANT SIZE
          0 VARIABLE FLAGS SIZE ALLOT

          : PRIMES
            FLAGS SIZE 1 FILL       \ All Marked as Primes before check
            0                       \ ( Initial_Total )
            SIZE  0                 \ Do All flags
            DO                      \ Ignore Total when marking )
                                    \ Ignore Fpos in the Loop )
              FLAGS  I  +  C@       \ ( Flag[index] )
              IF
                I DUP               \ ( index, index )
                +  3  +             \ ( Number )
                DUP  I  +           \ ( Number, next )
                BEGIN               \ Index+Prime is next Position to Eliminate
                  DUP SIZE <        \   Check IndexPrime (TOS) vs TableEnd (SIZE)
                WHILE               \ Mark LastPosition+Prime as NoPrime
                  0  OVER           \   ( Number, next, 0, next  )
                  FLAGS +           \   ( Number, next, FlagsPos )
                  C! OVER +         \   ( Number, next+prime     )
                REPEAT              \ Until marked all this Prime multiples
                DROP DROP           \ ( )
                1+                  \ ( New_Total )
              THEN
            LOOP                    \ Total is Primes processed up to now
            . ." primes in FLAGs"   \ Print Total, Done!
            ;

          : x10  FAST  CR 10  0  DO I  .  ."  ->"  PRIMES CR  LOOP  ." Done!"  SLOW CR  ;
```

***** RESULTS** of <u>10 workloads (FILL restored)</u> :

- Raw ACE measured values, 10 iteractions @3.25MHz:
  The original ROM = **382 sec** (@ 3.25MHz for a wider screen) ... 6.4 min
  Available TOOLrom = <u>298</u> sec (as above, with restored OVER) ... **5.0** min
  A perfect USERrom = 264 sec (No distorters.=> correct ACE) ... 4.4 min
- (on Basic/3.5MHz) = **4776 sec** (@ 3.5MHz, for a better speed) ...<u>79.6</u> min

Imagine the whole ACE restored, or rewritten to use the CPU stack as DataStk.
! You would get near 33x Basic. Better yet, imagine 4K RAM... We did, we do.

That secondary OVER

***** A Preliminary Study ***** ((OLD ENTRY -- Slightly edited))

```
FINAL Bench Timmings (10x iteractions)   secs    /Best   /Asm    /Ace
-----------------------------------------------------------------------
ACE    (4MHz RUN NoREMs) interp 4th       341.    24.4    50.1    1.00
ACE-aP (4MHz RUN NoREMs) interp 4th       218.    15.6    32.1    1.56
ACE1p  (4MHz RUN NoREMs) interp 4th       249.    17.8    36.6    1.37
-----------------------------------------------------------------------
```

- Times are for 4 MHz, loselly converted by Benchmark demand.
- **Ace-aP** is restored Primaries optimized (bigger) to load.(pre-patches)
- **ACE1p** restores only **OVER <** and **0**, for a free space supposed (was close)
- These values above where extrapolated after the following Critical Section:

=== Critical section ===

```
                               ACE                 OVER and 0
        BEGIN           \\   85               \\   85
         DUP (lit) <    \\  360+ 246 +1107    \\  360+ 246 + 861
        WHILE           \\  207               \\  207
         0 OVER         \\  246+1424          \\  189+ 575
         FLAGS +        \\  182+ 396          \\  182+ 396
         C! OVER +      \\  278+1424+ 396     \\  278+ 575 + 396
        REPEAT          \\  216               \\  216
                        -----------------   -----------------
                            = 6567  (1.00x)     = 4812  (1.36x)
 (Spectrum spends 4869s)      420s  (11.6x)       307s  (15.1x)
 Times above are for real Hardware, ACE=3.25MHz, Spectrum=3.5MHz
```

Workload represents 98% of SIEVE work, ~96%(?) of the whole Bench.
So yes...critical section is quite representative (so is conversion to secs).

```
 ACE +OVER +0                Was a preliminary study. (Later, we made OVER fit.)
-----------------------     On these, we also replaced '0' and '<' by a single '+!'
```

Shows that with 3 extra bytes for OVER, alone, Sieve would had been 15.0xBasic
And that, with an extra of 6bytes for a RSTs patch, would get a global 6% speedup.
RST patch is not advisable for compatibility reasons (we regret more the 6 bytes miss).

***** Final Results ***** ((New ENTRY - Using Weights3 instead the estimations above))
SIEVE indicated optimized code (space demanding) code as 1.73 faster. WEIGHT-3 shows 1.25.
Sieve previous results of 11.7x or 12.4x (using FILL, unnatural OR Primary) due the OVER word.
These were (surely) the source for a prudent but incorrect "10x" statement. Due the unnatural
Secondaries, Sieve later gives much higher values than Weights (see "Patches and Changes")

That reflects 'Weights' precision (for any FORTH) and its accurate conversion.
Then improved when FILL was removed from conversion. ALSO more fair to BASIC lacking FILL.

- ACE ROM shows **W =15.4x** , while **S =12.4x** (Sieve: Delivers a distorted 20% lower value)
- ACE+TOOL shows W =16.4x , while S =16.0x (Weights: OVER as prim, improved 6%. NOT 28%)
- ACE PRIV shows W =19.4x , while S =21.7x (Not fully Optimized... the gap will increase)

Weights gives uniform results by not depending from a word or two (as Sieve is, on the ACE)

Weights' is accurate: It is designed to **reflect** (*) FORTH real usage (even User Secondaries).
Sieve reflects a fixed problem, a limited scope (while Weights imposes close to none).
Both have their own problems, adding Z80 inadequacies (8080 heritage) here at play:

- Lack of indirection, short on registers (both of these improved on the Z80).
- And an Expensive second stack. With a solution not reaching the ACE.

Note: Refining conversion, Weights gives even more stable results (~1% error)
(This is done by removing FILL, on both ACE and BASIC. Also much fair!)

___
(*) Reflecting FORTH usage does depend from the frequencies data.
For our measures, we tried to obtain the most extended set possible.
Knowing well that set reflects a particular practice. Yet, broad enough.

1.5 Workloads Bench

More Accurate...
Forth allows more accurate measures. A synthetic method is natural.
We used common practice frequencies, from an initial paper on the subject.
Here are frequencies used on Weights2 (not on Weights3), to see how this works:

Words	Usage	Words	Usage	Words	Usage	Words	Usage	Words	Usage
(lit)	54.63	@	34.16	+	20.77	;s	20.02	?branch	15.93
col:	30.40	dup	14.27	OVER	9.53	c@	6.24	0=	5.79
c!	7.75	!	6.29	cells	3.17	=	2.69	1+	1.82
SWAP	3.24	+!	2.18	drop	2.67	execute	2.67	>R	2.17
I	2.09	R>	1.92	and	1.90	branch	1.58	user	1.30
>	0.91	or	0.82	?dup	0.73	*	0.51	mod	0.49
(loop)	0.44	2dup	0.24	NOTE: Argument copies (DUP and OVER) are commonly used					

The above table is shown only as a picture of the method. Also because useful for a quick usage.
Then, maybe still interesting. The new table used by WEIGHTS-3 is much more extensive, but also reflects 'advanced' practices. It's based on GForth authors generously shared data.
Versions 2 and 3 results are very similar as variations usually compensate each other.
We felt tempted to homogenise those slightly different sets (kept them as received).

WEIGHTS is nearly Parallel to SIEVE
A base of comparison must be chosen, an equivalence. First we used the +CODE patched ROM.
Later the TOOL patch as those SIEVE were no longer tainted with OVER. That way we got a good conversion factor, with (the taint removed was big) accurate results ... (Got ACE ~14.9x)

WEIGHTS is FORTH only
To cross with SIEVE results was a matter of respecting the absence of FILL on BASIC...
Thus comparing results WITHOUT that particular portion of code, on both BASIC and FORTH.

Cleaner Results ...
Better equivalence needs SIEVE without distortions (USER patch and a better emulator).
USER patch eliminates *both* ACE distorters of the SIEVE (no longer just OVER), allowing a more accurate equivalency of results. Immediately we noticed a small 4% change, resulting from a distortion smaller than the huge distortion of OVER. THE Final Result: **ACE = 15.4x**

Solid results ...
To mention "The SIEVE effect", one that only Forth (without distorters) may reveal :
- The smaller the set of OPs, the bigger the variations .

Also to mention that SIEVE favours common 16bit programming of 8 bits computers, while Weights3 reflects high-end programming instead. Sieve is still useful, when not tainted.

Conclusions:
* The full spectrum of results more stable than ever, also correcting the /ASM ratio. All ratios now make more sense. (Note: Weights3 lowered OVER argument copy frequency.)
* Results of different patches (when immune to distortions) go parallel with SIEVE. An apparent exception is the table reference CPUstk. These a mix with one value referring ZIP, and the other the value of FIG-1.1 (of which we have no sources). Its indicative data.
* Beyond 'Weights' accuracy, conversion allows a good degree on comparison with other languages.

News:
We made the internal values of WEIGHTS more 'readable'. Now showing an instruction workload average, instead a fraction of the full-set total.
(It's the same thing, internally. Now expressed more meaningfully)

Clean Evaluation

Understanding the whole
In spite of how 'clean' the results, the point of ACE-Forth is not (not really) its speed.

The main point is being a structured language, compiling without the need of an host computer.
This was achieved on an 8bits micro-computer with nearly no RAM, where intended 4K was reduced to 1K due an older design, not adapted to Dynamic Ram (4k for the price of 1k).

Reminded that there's more than (important) raw speed let us clarify a few details there:

ROM vs Implementation vs Language Speeds

* As seen on the Benchmarks table, ROM speed was 15.4x Basic
  This in spite of a few Word Operators turned Secondary Words.
  The TOOL Rom Patch corrected 'OVER', Speed is now 16.4x Basic

* The USER Rom patch also corrected '<'(and more), Speed is now 17.9x Basic
  With these Operators no longer disturbing measures, we now 'know' the Implementation Speed.
  DO WE?!? ... At least we finally have a clean basis to get a parallel with Sieve values.

* Unlike extremely limited and fixed Batch Engines, Forth words can be altered.on a fixed implementation. This is specially true when a Rom is forced to deliver Primary words as Secondaries, or when important Primary Words are removed to be built later.
  Doing so, the limit is the Architectural Implementation.
  PRIV Rom Patch is close to that limit, ~20x Basic

* On development, the ACE Architecture most certainly was to change. Demanding a rewrite.
  Somehow that did not happened, though a rewrite month loss would ease ACE 'inovations'.
  MMS-Forth for C/PM also started with SoftStackwith similar results. Changed to CPUstack.
  FIG 1.0 for the 8080 most likely started that way, being a reference before that change.
  Its speed is ~33x Basic, as MMS-Forth, and the ACE2 that would demand a month(?)

As we can see, there are various speeds for Indirect Forth.
Relevant to the Jupiter ACE are a Rom-Speed of 15.4xBasic, and an Architectural-Speed of 20x achieved by replacing some Words, and adding the missing.

This, because there WAS a real language available for cheap computers, no longer toys.
Nor damaging the perspectives of those interested on the theme, as a legend stated:

> *"It is practically impossible to teach good programming*
> *to students that have had a prior exposure to BASIC:*
> *As potential programmers they are mentally mutilated*
> *beyond hope of regeneration."* ~ Edsger W. Dijkstra

Speed could vary from good to excellent.
But the effects on clarity of sight were always excellent.
<u>Such is the memory of whom has known ACE-Forth</u>, in spite of the Jupiter final Hardware.

What about today?

> While there was a push on dependency (of libraries)...
> While there there is an invitatin for the easy but slow...
> The lesson remains, by compilers that do not hide the stack.

You may call it educative, perspectives widening eye-opening... thus anachronic.
It's your choice, our choice and exercise. Period!

Some Stress tests

Speed is not all there is. While important, it's a detail.
With that reminded, there is a better 'Benchmark' deserving that qualification:
Simple and short, we've used it plenty by the 80's, while playing with a similar

Also an impossible bench until asked to stop (crashing the University mainframe). Oops!
Both recursive, we shared both on the Jupiter ACE forum, somewhere between 2007..2011:

Since not at hand, here rebuild FIB using the "Ultimate Forth Bench" version.
Not a real Bench set, 2 algorithms DO qualify: Sieve and Recursive Fibonacci
Many 'benchs' there are stress tests, and 2 of them measure used CPU Ops, not Forth.
New 'Nesting' does its job well, cleaner than FIB, but not a versatile... Nor a bench.

The OTHER Bench: Recursive FIB (more than it seems)

```
\ Recursive Fibonacci IS a versatily Bench (small, Arg based, fairly complete)
\ This may be the most useful Bench, small and fairly complete. (but in 1982, excluded BASIC)
\ Apparently it mainly tests Nesting & R-Stack (depth and access)... Not only, as follows:
\ Also tests D-Stk:  DUP (twice as due) SWAP (LIT) < IF 1- 2- + (only drop is absent)
: Rfib ( n1 -- n2 )               \ Ace-Def accept non closed on itself
    dup 2 < if  drop 1 exit then  \ Terminator (argument test)... done?
    dup  1- fib1                  \ Recurse Previous value
    swap 2- fib1  + ;             \ Recurse Preceding and sum... done!
: doRfib  (--)                    \ Keep result inside Integer 16bits range
    21 Rfib  U.  ;                \ Equivalent to 20 0 DO RFib Loop. Shows last value.
: xRfib ( scale -- )              \ Need a Multiplier ?
    0 DO doRFib LOOP ;            \ Try Scale = 5x with   FAST 5 xRFib
```

Stress tests deserving a mention

```
\ Nesting                         \ Stress Tests, clean, not as versatile as RFIB
\ Not examined here, some progress arithmetically. Other exponentially, as RFIB mentioned above

\ D-Stk OPS     \ Not a Bench. A useful Data-Stack stress test,(not limited to DUP DROP)
DECIMAL         \ just in case
: Stack  -1 20000 0 DO            \ ( n --n ) ... 20000x5 Ops = 100.000 per round
    DUP OVER  ROT DROP DROP       \ Leaves a seed on Data-stk for the next loop
  LOOP DROP ;                     \ Initial -1 was a dummy value, dropped at the end
: StkTest ( n --)  0 DO  Stack  LOOP ;
\ Test  with    FAST 10 StkTest   to run a Million Stack Operations (10 x100.000)

\ Division '/' stress test.replaces "+-*/" : All are submerged by "/", all are irrelevant
        \ NOT NEEDED, here, to add System Stack missing words defined for the ACE.
        \ : SP!  DUP + HERE + 12 + 15419 ! ;    \ Set StackPointer SP (ACE only)
        \ : SP@  15419 @ HERE - 12 - ;          \ Get StackPointer SP (ACE only)
        \ : SP_  0  SP! ;                       \ Stack reset, non standard
: Division  ." Start"             \ The usually most affected Arithmetic Operator (Runs clean)
    1000  0 DO                    \ Inner scale is 1.000x, enough for that Operator
      2570 165 / drop             \ Representative Magic numbers (12bits/8bits)
    LOOP ." Ended" ;              \ Above (literals+drop+loop) are meaningless
: DivTest ( n --)  0 DO  Division  LOOP ;
\ Test with   FAST 10 DIVTEST   to run 10.000 Div's (more than enough).
```

NOTE: When Division is needed, we advise the use of U/MOD on Unsigned Arithmetic (faster).

ALSO: To play with the "Ultimate" set (it's their stated purpose) you'll need these Hacks

```
        : \  query CR ; immediate              : [CHAR]  [ find ASCII , ] ; IMMEDIATE
        CREATE R@      find I @   here 2- !     CREATE '   find FIND @   here 2- !

        : CODE  create here  dup 2- !  ;         : HEX  [  decimal  ]  16 base c! ;
        HEX CODE  +!  d5df , e1df , 234e , 2b46 , 09eb , 73eb , 7223 , e9fd , DECIMAL
```

On Hidden Engines

Old Code and Benchmarks and BASIC09 <by James Jones, May/June 2019, on VintageIsThenewOld.com>
"Recently a gentleman asked me whether I have a CoCo 3 (more formally, a TRS-80 Color Computer 3). He was curious about whether I could run a couple of benchmarks both in Color BASIC and in BASIC09.
(...)

```
PROCEDURE sieve
        BASE 0
        DIM i,k,prime,count:INTEGER; flags(8191):BOOLEAN
        PRINT DATE$
        PRINT "Only one iteration"
        FOR i:=0 TO SIZE(flags)-1
          flags(i):=TRUE
        NEXT i
        count:=0
        FOR i:=0 TO SIZE(flags)-1
          IF flags(i) THEN
            prime:=i+i+3
            FOR k:=i+prime TO SIZE(flags)-1 STEP prime
              flags(k):=FALSE
            NEXT k
            count:=count+1
          ENDIF
        NEXT i
        PRINT count;" primes"
        PRINT DATE$
```

(...)

> *<u>The result</u>:*
> *ten iterations took between 114 and 115 seconds, though the range has to be two seconds (in the best case, the starting time is just before the second after what's shown and the ending time is barely past what's show, and the worst case is the other way around)."*
> <End-quote>

Some Notes on this excellent 'Pascalic' renamed (functions adapted):
By that time, we made that request to a gentleman on Facebook (contact was lost).
Above Procedure was for one iteration, supposing 'Basic'. Later 10x was not shown.
These results were quite unexpected. Are also very interesting for various reasons.

*Facts are: 6809/2MHz is **1.8x** faster than a Z80/4MHz but Coco3 is **1.61x** of a Z80/4MHz*
*Therefore a Coco 6809/1.85MHz is ... **1.98x** a Jupiter Z80/3.25. This meaning that those **115sec** measured on the (twice faster) Coco3 do parallel the **225secs** of our ACE-PRIV patch.*
ACE-PRIV patch optimizes stack usage, closer to the expected on a 2 Stacks 6089.
So close values are a coincidence, yes, but with reasons we may understand.

1st reason: <u>So close results</u> enforces suspicion that BASIC09 was (by the 80's) a threaded language as BBC BASIC was. That's not BASIC on steroids, compiled, but a good Pascalic.
Does it use a Dispatcher equivalent to the one of Indirect Forth? ... It's most likely.
Under the marketing name we find a a Pascal-like.language, a different structure.
With a few procedures to justify the name (as BBC Basic), everyone benefited.
On the 90's Borland Turbo-Basic really was a C engine (plus those cosmetics).
And Visual-Basic, a reface of Object-Pascal (after losing against TPascal).

2nd reason: The ACE-Forth older architecture compensates the extra delays of argument managing. demanded by Pascal-like implementations (as exhibited on PROCEDURE)... Thus, similar results.
To Notice: Forth on the 6809 would be faster than a Pascal-like translation. Yet, not using the 'magic' name would hardly be accepted and sold. A Pascal-like could be renamed. It was.
There were several similar "structured Basic"s... No longer that, the name ensured success.

Chapter 2 - Clarifications

Some quotes:
"It is practically impossible to teach good programming to students that have had a prior exposure to BASIC : As potential programmers they are mentally mutilated beyond hope of regeneration."
~ Edsger W. Dijkstra

"*What I liked about it was the way you could REDEFINE words *without* having to FORGET everything defined afterwards. Do any other FORTHs work like that?*" ~ Anthony Hegedus

"*I loved my ACE (...) it was a mystery (how did it tick so fast?). ... Later found [REDEFINE] to be unusual, though it seemed natural. Unfortunately, I've only enjoyed 1k... Actually, 900 bytes.* ~ DuLac

Sharing some Reflections:
"Languages hide their details from the student and the programmer."
"FORTH joins language with code. Two stacks allow to keep the 'essence'."

"The stack is not to be hidden, but to be used for what it is."
"The stack is an essential mechanism for both control and storage."
"FORTH allows complete control to the programmer, as Assembler does."
"Temporary local variables, kept on the stack, are very close to Assembler."

"The KISS rule (Keep it Simple) and the needed Hiding, are NOT equivalent."
"Simplicity is on the management of complexity, not on its absence."

Clarifying some Myths:

* *"Forth is interpreted language"*... NOT! (Only user interaction is)
  It's a compiled Language, How it is compiled depends on implementation.
  Forth original method is Threaded Compilation, of pointers run by a dispatcher.
  (Treading can be Subroutine based.) Using a dispatcher is the equivocation source, though much different than using an interpreter. There is a translation action on compiling, sometimes lazily referenced as 'interpretation' (a flawed designation.)

* *“Forth code is obscure”*... NOT! (The statement should had been “lesser known”)
  It's less hard to read than Assembler or a BASIC batch. Everything is named, identified.
  As with any real language, bad practices may be used. There too, these are allowed.
  Ever tried to learn a Latin Language? Or English, a foreign reversing language.
  We are all sensible to the effects of our ignorance. We even forget Gibberish is possible on any language (without practice we may not distinguish).

* *“Forth is its stack usage”* ... NOT! (It is just not hidden)
  Assembler is not the stack, every language use the stack (hiding it).
  Reverse Notation is a result of not hiding the stack (to avoid management delays).
  So it ‘works’ as an Assembler, without extra-copies to satisfy arguments positions.
  That is what the stack means: It's an argument passing mechanism (just not hidden).

* *“Forth lacks needed Local variables”* ... NOT! (They just not need to be declared)
  Practice suggest the use of the Return Stack, limited to be fast, for that purpose.
  It's considered ‘advanced’ because some care is needed, thus usually not mentioned.

2.1 FORTH vs BASIC ?

We should not compare different natures! Maybe a few characteristic references shared.
Comparing, may be a desire. Here based on availability and fitting in a very small ROM.

Altair BASIC batch engine has received a few useful routines (requested by Altair).
Still, some of the reasons NOT enjoy Batch Engines, is the crudeness of line numbers (the absence of structure language). Also the "one command does many things" strategy. Line numbers promoted "Spaghetti Code", induced Code obscurity. Not even ASM had those.

BASIC was never a language, but a script of command routines (then, many 'interests' involved). Yet, there were good threaded implementations 'named' BASIC. Named to satisfy a demand for that reassuring name, mantra-like. For that demanded supposition of simplicity, many competent people later tried to fix the 'basic' delusion, but forced to show its (limited set) of script commands.

Note: C/Pascal compilers (structured) with added 'recognisable' BASIC routines" should *NOT be confused with the original script application: This was crude, line numbered, but available. It allowed some programming. The seductive name survived,but alone.*

That trick credited (!) the ilusion: Were not the same, just kept the name.
Honourable mention goes to BBC 'Basic' and others that followed, Coco 'Basic09'.
While the difference was clear, and fast, the equivocation was easy, even desired.
(Some implements could be called 'Forscal', ie, Pascal-like with Forth-like engines)

On 'Legibility' vs 'Obscurity', a silly example with very few elements:

```
10 PRINT "is Great as " ; RETURN
30 PRINT "Copying BASIC " ; RETURN
40 PRINT "Soda " RETURN
50 PRINT "BASIC Programming " ; RETURN
70 GOSUB 30 ; GOSUB 10 ; GOSUB 40 ; GOSUB 10 ; RETURN
80 GOSUB 50 ; GOSUB 10 ; GOSUB 30 ; GOSUB 10 ; RETURN
90 GOSUB 70 ; GOSUB 80 ; RETURN
```

Note: run 70 would result in: "Copying BASIC is Great! as SODA is Great!"
run 80 would output: "BASIC Programming is Great! As Copying BASIC is Great!"
On some implementations (batch not yet doing real calls) RUN 90 would crash.

In FORTH the same could be a set of strings. (The silly example bellow shows the mechanics).
Following the BASIC example above, a procedural version for the script above could be

```
: ==   ."  is Great as" ;        : 4th  ." FORTH Programming " ;
: 8bit  ." 8bit" ;               : Cream ."  Ice_Cream " ;
: !bits!  4th == 8bit  ;         : !Cream!   4th == Cream ;
: BOTH  CR !bits! CR !Cream!  ;
```

Where:
(:) means start of an instruction(routine), name following
(.") means print the following string, until a (") marks its end.
(;) closes the procedure definition, ending it similar to " ending a string.

Once a Forth Word is correct... It's done!
Yet, we can later change it. On the ACE we can actualise a previous build to a new version.
! Beware changes do not conflict with previous 'expectations' (i.e. attempted usage).
! REDEFINE benefits imply responsibility (what you tell may not be what you want).

Incremental compilation! *(Voyager Speed and Size, beyond Pluto)*
FORTH is about flexibility, not about format. And Efficiency (as speed and size).
It's legible by your design (the example shown is silly). It's programmer's wise.
? Wondering why FORTH was/is so used by NASA? Not due this, there is a bit more to it.
! At a time, FORTH was considered their secret tool (Hints: Extensibility, Self-Run data).

2.2 A Human Interface

In the beginning programming was done by hand, byte by byte.
Trough the 70's, a main concern was to get a practical interface. That was an hard problem.
On the 8bits arena, small batch engines replaced host languages as a poisoned bliss (see Dijkstra)
With the 90's we got used to advanced CPUs, huge memory pools of RAM and storage,
to fast devices we buy in a store. And fortunes were made trickering everyone.

Starting the 80's, micros were an amazement. A time of discovery and accessibility, with the micro CPUs 64k barrier, usually 48K, and even small RAMs... from 1k bytes TO 16K.
16 K was a huge investment, eased with dynamic RAMs (for which the Z80 was ready).

Is it a bird? Is it a rocket? ...
Back to middle 70's, small systems had not enough RAM for compilers. So, an application batch was well received, Altair suggested useful routines to be included to his contracted 'obscure' students out-of-place on MIT, eager to play 'enterprise' away. BASIC teaching set become that interface tool.
It seemed much, no longer having to enter a Program in Assembler, one byte after another.
Fortunes are made exploiting ignorance, selling shiny glass for centuries. Still do!

Long before (not an hobby) FORTH become a tool for everything on its very practical evolution. Not a batch. Able, practical and flexible, it was a tool one could use and trust. Small enough for a compiler, and interactive. Without pomp, it was the much needed interface to 8/16 bit CPUs.

Sadly, due 8080 CPU's limitations the Z80 shows, it was not a good CPU to run FORTH.
The 6502, harder to program, was better. Perfect, was the 6809 (killed by Motorola).
Much more acceptable was (later) the 8086, even if much less than the 68000 series).

... It's the FORTH Language! (not just a set of commands)
Forth was not designed. It has grown atomic and self-consistent. Forth 'happened'. HA!
Has grown both a language and an OS. On board CPUs, on Chip CPUs. Was it a philosophy?
That too, maybe... depends on you. Organically elegant, it has no syntax (nor limits?).

All its elasticity comes from there, at a price. That price is dealing with the stack other languages simply hide. Not totally true: Its a choice (for efficiency) not to use local variables. These CAN be used, reserving space on either stacks.

Efficiency demanded to manually control the stack and hide the mentioned elegance with
stack management invocations. Much faster than declared data (unless DataStack is slow).
Faster than referenced local variables. (We can still access 3 locals with I, I'and J).

Forth is NOT that pervasive use of stack. Forth is about Expansibility, about Types creation.
Also about keeping the responsible, as responsible: Demanding them to know 'the' problem!
To know its essence! Also (due stack management) more "High-Level Assembler" than 'C'.
(A reason behind the suggestion to factorise Word actions and to keep words short.)
For those reasons, Forth reveals whom uses it: It's an enhancer! Real, and honest.

... And it is NOT a Language (not 'just' a language)
It's more like an Assembler, for a Language. NOT a Batch of commands (sold as a language).
ANSI has tried hard to give FORTH elements of limitation, so dear to standard minds who need definitions and limits, to be pleased. Some structural elements of limitation were ADDED (as extensions) to satisfy the need for standardisation (useful, no one denies).

An immediate example may be, the CASE structure, useful for readability.
But also hiding identification of alternatives that MIGHT be more adequate than
the inefficient sequence of IFs, all 'languages' hide with pleasant CASE syntax.
(Off-topic: Which ring bells on other matters, as parallels can be found easily.)

___
Some call FORTH a religion, too... (The term usually a church and a dogma).
The above was stated with a laugh! Some, maybe will understand How (not Why).

2.3 Or an High-Level ASM?

? Being an high-level language, why so low ?
It's a question of efficiency, keeping it close to assembler.
The programmer manages the arguments passed to routines. As well as temporary local data.
Other Languages hide how arguments are managed, as a service (an expensive one).

? What then? It's ASM or is it High-level ?
It's conceptually efficient, thus both in that very short package.
So we may say to be High Level, very close to the CPU. Naturally both, also naturally recursive.
And that's "conceptually efficient".

? How is that ?
Being close to the CPU and, in it's traditional form interpreted (even if partially compiled)
the administration of the arguments is a responsibility of the programmer (as in ASM).

? An High-level Assembler?
As C was called on its first years ?!? It's similar.
Bear in mind C was a native mode compiler, while FORTH was a uCode (thus a threaded 'thing').
It did not followed the trends of the previous decades. That allowed it to be practical and sane.

? High is OK. But an Assembler?!?
Each word (or function, if you like) is autonomously executed ... as is.
In short, there's no linker. Any definition being immediately ready to be used (or tested).
    If it's right is right. If not, it's not. Everything must fit as it comes, kept simple
    (with no later 'rules')

? But done manualy is it not... inconvenient?
It's an extra work. Due manual arguments management, 'words' are only locally stack dependent.
Limited to essential actions, 'Words' are easy to deal with if rational, hard if not.
In the end, it results well (even 'convenient'):
- Each Word (function) can be tested individualy. Inner details can later be ignored.
- When a new 'word' is done, it's done. !!! THAT is extremely advantageous!

In conclusion:
The end result being a very compact language, where the words can be used as functions,
procedures or Macros. When correct, are done! And available as if part of the language.
It's up to the programmer. Its absence of barriers confuses more than Args management.
Everything can be done without reassuring limitations. Freedom means responsibility.

Do remember:
This may be our best advice: The stack is not a replacement to variables.
It may be felt as due. That is a mistake, that is not its role.
A Word has its local data space, shared trough the stack.

Its data are local variables, being more as values on CPU registers.
You are not forced to declare them, not unless you want or need.
Arguments and temporary values, both have a different nature,
not to be confused... unless you choose to make a mess.

Never force! Use your discernment. Know the nature of what you deal with.
You can, you should know better than doing random programming.;)
<u>It allows you to decide</u>. For you to manage that freedom.
(That can be scary if trained to follow limitations.)

Freedom has the power of choices. To be deserved.
If not, will distract us.all... into its opposite.

Chapter 3 - Programming Tips

3.1 For a Clean Start

The only strangeness of FORTH ... May be its ASM-like stack, for Temps&Args.
Every Language hide the Stack, except ASM and FORTH. Its usage satisfies several goals:
Call/Return, Argument passing (as 'Temps'). Forth simplifies that multiple usage with 2 stacks.

There's a popular belief...
that "Forth main characteristic is Stack management" NOT!, that's just a mechanism. For efficiency.
That misunderstanding has a reason: Stack management is usually hidden, to 'simplify' being used
for different purposes. Forth simply assumed another stack was needed (LISP would benefit that).

1st - Before to start :
* You may wish to get an emulator (if you do not have one already).
  One good enough, available to all OSs running DOSbox, is ACE32.exe (93K EXE + 8K ROM.file)
  Contra: It interferes with ROM (on boot,to enforce a 32K space) disabling patch of Z80 RSTs
* You may wish to print the best introduction to the ACE (by the ACE author): The Manual (PDF).

2nd - A clear approach :
a) Recognise FORTH to be "syntax free" (except paired Control and Type-Definer Operators).
   This is related with use of reversed notation (as commonly stated), as a direct argument.
   passing. ASM-like. As reminded, ASM does not hides its stack either: CPUs work that way.
b) Interesting about Forth, beside 2 separate stacks, is the capability to create commands
   in place of routines: 'Words' extending the language with/for flexibility and power.
c) But also (!) the ability to create new Data types and Code classes (long before OOP).

3th - Two important, very simple TIPs :
#1 We would suggest to use variables as with any other language.
   The stack is used for parameters passing, and for quick work. ONLY!
   Learn to distinguish both, not be confused. Nor by the flexibility it offers.
#2 Also remind the ReturnStk IS the Forth equivalent of ASM-stack: Also temporary storage.
   Remind I, I' and J can be used for consultation of values temporarily 'pushed' there.
   That you must 'clean' any such 'pushes' before ending the Word (it's system, not yours).

A final note, to dive into ACE-Forth (maybe the easiest introduction to Forth, small&usable)
First: Read and exercise the ACE Manual.contents. It's a very good introduction to Forth.
Then: On this docs you'll find some missing words to load. Mostly missing Primaries.
Some on snapshots, of emulation 'state', to a quick use (as a tape load).

Overlooked, but most important: Define the HEX word (so not to trash the FORTH Voc Header).
This way, experimenting will be easier and faster. With the CODE word you'll be able to
create your own expanded ACE... Then closer to the intended implementation of Forth!

With these short advises, and the introduction of the ACE manual, the Forth road is easier.
Have a profitable start: May improve it using the ACE_TOOL ROM, if desired (GPL-3 licence).
What will follow is only up to you (an advise always valid as some decades ago):
- From the degree of your own curiosity (multiplied by your own motivation).

___
(?) Why not to restore the internal D-push, D-pop? For compatibility reasons.
Also, ACE32.exe alters the ROM image to force available memory to a 32k limit
((Affected bytes (by the ACE32 dirty patch) are the 3 code bytes at $002A
making any change of RST28 content largely incompatible with ACE32.exe))

3.2 DEFINER/COMPILER

Before going forth (further) remind this:
1) ACE-Forth decompiles User Secondary Words, as long built with words it recognizes.
2) ALL 'Words' are compiled, whatever its type (On Forth, all Types are virtual classes)

There are 4 initial Hard coded Types. As such, they are NOT decompilable:
- Primitive words \ Fundamental Forth operations. Usually in ROM (unless user built)
- VARIABLEs \ Place its address on stack, usually to fetch store its value
- CONSTANTs \ Faster as fetch only (by itself), its address is shortcut

Then there are User-Built Secondary Words. These ARE decompilable:
- Secondaries, Definers, Compilers... are Head references followed by a list of Forth words.

Creating words makes everything more interesting. Better, the ACE will allow to alter them all. Either common ':' colon words (most of user new definitions), either classes (DEFINER types), and even new programming structures (COMPILER directives)... to be invoked on compilation.

SECONDARY words: \ Example being MSG1 to display a repeated message
- COLON (:) allows adding words to a new word. (First creating its header reference.)

DEFINER words: \ Example is ARRAY <name>, capable of transforming index into location
- Create a reference (Header) to an object class (as a Pascal Type) with a shared behaviour.
- That new class (Type) has EDITable Secondary Code. Data referenced not EDITable:
  An object type usually consist of data, not of code (related code is the definition)
- Words a definer create exhibit two behaviours:
  To build, at Defining time. Be a Type, at Run time.
  (A Type is a Class with a shared single behaviour.)

COMPILER words: \ As IF-ELSE-THEN A user built is CASE (whatever implementation wished)
- Create a new structural programming directive, as a Secondary Word (EDITable and LISTable).
- The created word is IMMEDIATE. As DEFINER above, exhibits 2 behaviours (on build, on run).
  A manager, at compile time. An action at Run time. DOES NOT create a class of words.
  Instead, it builds structural words (on the editor)... *and* its run-time action.

While COLON (':') is the standard building word (similar to functions in C), DEFINER and COMPILER differ from standard Forth CREATE BUILD-DOES> to achieve these simple goals:
a) Both are editable and redefinable, as COLON is.
b) DEFINER is used to build Data structures (invoked on User Interpret stage, on Run-Time)
   The defining word-type delivering their children a common behaviour (code)
   Please note that those children, are just data, not code to be edited.
c) COMPILER is used to build Compile-only words (directives as IF-ELSE-THEN, on Compile-Time)
   It creates word-directives also with a double behaviour according to compiling time
   or to run time. NOTE: Compilation directives create no children (act immediatly)

Again, what where the goals?
Primarily: To distinguish their nature... So to allow decompiling, thus re-Edit. (Ha!)
Secondarily: Simplify creation of compiling directives with COMPILER (also Redefinable).

((There's no need (nor will) to dive on the inner details of a multi-level tool.
3 stages of dual action (total is not 6 but 5 states. Then there's the advanced COMPILER word with 2 stages only... Tricky? But to simplify usage while giving an extra power.
- *As stated, there's no need (nor will) to read the details, nor to 'justify' them.*
- *Life is where "the Simple" promotes the coherence needed to allow variety and complexity".*
  'The Hard' is to build an added dimension of meanings, keeping it simple and easy to use.
  Kind of building this Universe, so people can sit in front of a Computer. Still there?))

 # Listened another day, the amazement of seeing a pig riding a bicycle: Was that amazing?
It was a triple sin! A waste of time, a useless endeavour. But mainly, it annoyed the pig.
... Everything has its own place and time. So, relax! ... Enjoy this unusual ride.

3.3 Using VOCs Tree

ACE VOCs are not the standard, closed VOCs. They present a compromise, open Vocs.
This serves tree travelling well. Sadly, code is incomplete (Scope will not always function).

Knowing Search is ONLY for reaching other VOCs, the way to use VOCs becomes clear:
 * Use a flat system. I.E. all VOCs over root. (Next, we will ease that strategy)
We can then quickly alter context, in order to find a particular Word (to be used).
Will avoid misfinding (or no findings at all) when defining a new word. We reach them.
To respect disassembly, we'll use COMPILER to store Context changes (for EDIT or LIST).
(That's another use of that non-standard Forth addition. ACE-Forth is well integrated).

Quickly Visible, and easily Reachable on Build-Time:
A useful trick is to build VOCS, named within colons to be seen easily (not its pair).
Use dedicated context changing words, pairing each VOc (named with square brackets).
Note: We have tried several several options (found these choices to be ‘cleaner’).

An example of a new vocabulary, accessible by invocations (on any state):

```
FORTH DEFINITIONS
VOCABULARY :UTILS:                        \\ name is very visible on Vocabulary
0 COMPILER [UTILS] FORTH :UTILS: RUNS>  ;   \\ Name reminds immediacy of the word
```

These names will not be easy to forget. Nor by us, nor by the decompiling system.
Note: Colon words (:) made IMMEDIATE are not compiled, COMPILER made words are immediate
Using COMPILER solves the invocation problem, immediate but no longer lost for EDIT.

This UTILS vocabulary was an example of a pattern to follow.
Use it for all new Vocabularies. A Vocabulary invocation (on compilation) change CONTEXT.
After benefiting from that change, no longer needed, a return to FORTH context is needed.

We also need a prior, common compile-time return to FORTH context (for consistency) :

```
FORTH DEFINITIONS
0 COMPILER [FORTH] FORTH RUNS> ;       \\ Should be declared BEFORE the above example !
```

It ends a compilation-time context change. (Be consistent: Make it the first word defined)

A usage example of invoking a word “far away”, going&returning...will be simply:

```
\\  [vocname] word invoked [FORTH] ...  \\ change context, invoke, restore context
UTIL 16 wARRAY FORTH        \\ On the main Library, creates a 16 integer indexed array.
```

Using DEFINER types inside a Vocabulary is similar, but built on interpreter-time:

```
\\  :vocname: invoked definer FORTH ... \\ change context, invoke, restore context
: SP_  [UTIL] 0 SP! [FORTH] ( no further action but do restore CONTEXT anyway) ;
```

It's no longer a problem if we are in interpreter or compiling modes: We have both accesses.
Only the psychological problem of MAYBE forget were we are, or were the new definitions go.
Thus an advice: Never lose track of were definitions go!   You change CONTEXT, remind it !

Conclusion:
!Know what happens! That avoids unexpected rubbish (on any Language, also Forth):
!!! FORTH does what you tell him to, it does not guess for you!!!
Flexibility increases opportunitiesboth to build or to crash.

So, believe nothing... Try to know HOW, as well as WHAT you want (also valid on other arenas).
... BE AWARE of where you are and do... Or you'll be fooled (and crash).

P.S. NOTE:
 With most retro machines, you will not use BASIC, because code is obscure (and slow).
 Not so with ACE-FORTH. Sadly not used, not because old. Because the Jupiter has 1K.
 Fortunately, there are emulators. There, the ACE runs a completely different game.

3.4 Using STRINGs

A minor, overlooked problem :
Managing Strings is relatively important, quasi-simple to restore the missing tools.
(FLOATS were considered more important to do serious work, thus kept.)

Strings did become another miss: When compiled, they are exceptions to disassembly.
Good examples are the first words one learns: **."** print word and Comment parenthesis.
Disassembly need to know each, to deal with the exceptional extraction of data compiled.

How does FORTH works with STRINGs? (i.e. beyond writing with the embedded text **."** command)
In a peculiar but very efficient way. It's implemented as in Pascal: A counter followed by text.
HOWEVER, more is available: Two work parameters describe an inner string. A link and a length.
Enough to describe and manage any text. These are descriptors, to work on storage elsewhere.
Pascal can, maybe should, formalize those two work parameters usage when working on Text.

(After ACE manual) we get text from keyboard, using the word WORD (no pun intended).
WORD has a similar job as NUMBER. Both work on the inputline (not on code written).
The PAD gets a text this way: D*elimiterByte WORD <inputline> <delimiter-char>*
As in the Jupiter ACE manual: ASCII " WORD this is a text " ... .

WARNING - Found the first bug of the ACE: WORD does not allow empty text (we have to zero it!)
PAD is correctly filled with spaces. But sized as 30 (instead a zero). No ROMspace, no check.

```
: COUNT   DUP  1+  SWAP  C@ ;   \ Extract Content Parameters... Weird, but versatile
: INPUT"  ascii " word ;        \ Get User Text into the PAD, using '"' as terminator.
: TEXT?   COUNT  TYPE  ;              \ T? Utility: Examine Text on Array (Consume Params)
: STR?    2DUP SWAP U. U.  type;      \ S? On both utilities: Use 2DUP to preserve Params.
```

COUNT is the word that reads a CCArray (an addr), then giving us its Text Parameters pair.
As Text Parameters point Strings, PAD area is enough for temporary Str management... How?:

PAD serves as <u>**C**ounted**C**har**A**rray</u> Buffer. But we <u>may</u> need storage to use later, may need <u>other</u>
CCArrays available as Variables. Before we do so, we remind the two ways to deal with a text:

* Array 'storage' : The Counted Chars Array (Addr), pointed as any variable is.
* Were to start & for how long : The Work 'description' pair (StartAddr Lenght)

<u>As long as we distinguish the Object from its work Parameters</u>, we can do anything.
We will call '**s**tring' to <u>working parameters</u> (the add+lenght pair). Prefixed by '**s**'.
We will call '**t**ext' to <u>contents stored</u> (a Counted Array). These prefixed with a '**t**'.

```
definer tArray ( byte -- )      \\ warning: max is 253, not tested
  dup c, 0 c, allot             \\ reserve limit, size, space reserved
does>  1+  ;                    \\ ( -- addr )
```

We have included the maximum size of a declared string behind the ChArray itself.
It's an exercise of safety (info on Array max size available) to use later. It pays.
Before using Strings, an example of getting the (content) Parameters for a clean text:

The **-TRAILING** word does not alter the text but the work parameters of a string in it.

INPUT" <u>ABCD</u> " PAD Text? \\ We get "<u>ABCD</u> " OK
PAD COUNT -Trailing TYPE \\ We get "<u>ABCD</u>" OK

Note that **-**Trailing works with parameters. The Text on PAD is untouched.

We could had used a shortcut, useful if repeatedly using the PAD :

```
: sPad  PAD COUNT ;             \\ Leaves PAD str parameters instead PAD address
```

sPad is a shorter name than a more legible PadStr. We assume the 's' prefix to be enough.

For flexibility, TYPE works with parameters. Working with them allows to get <u>any</u> job done.
We may latter change the original string (reason why we store MAX size on 'our' string class).

Two Strings tools

The following words were named carefully (coherence eases later work).
Thus we use tArray instead ccString to keep our nomenclature unequivocal.

Example : Start saving introduced text on storage arrays previously defined.

```
13 tArray MyT13          \ Builds a small Text Array of 13 chars, maximum
8  tArray MyT08          \ Now a smaller one, of 8 chars maximum. Beware sizes!
```

We will need a word tool allowing to move our example on PAD, or our selection.
It will be used as this, maybe after INPUT~ (as PAD content is very labile)

```
Pad count -Trailing   MyT13 >counted   \\ >Counted is not defined yet
```

Our attention goes to rationalise and detect useful actions, Moore's 'factoring'.
Wonder the origin of his 'factoring' words advise? Joking with a pun for 'factory'.
This demands a bit of work, by the factories we are. It pays latter: Tools available.
We did different implementations, and found factorization was already done: It's cMOVE

The following storage Tools, both alter storage: (maybe even on the middle of its Text)

```
: Txt2T  ( OrigArray DestArray -- )        \\ Simple direct copy between similars
  over c@ 1+ cMOVE ;                       \\ !!! Safe Destination size test is missing
```

And its pair, originally named >COUNTED ... A nice name, but we may need 2 kinds of copies.

```
: Str2T  ( OrigStr length Destarray -- )   \\ ChArray(Params)... To Counted CharArray
  >R dup I c!   \ OrigStr len              RStk: Darray(with size)
  R> 1+ swap    \ OrigStr DDestArray len   RStk: ..
  cMOVE ;                                  \\ !!! Safe size test is missing here too
```

After the usual [*Input" A test   "*] we will test our tools:

```
PAD    count type                  \\ check tArray (system, buffer)
MyT13  count type                  \\ check tArray (user defined, on top of page)
PAD  MyT13  Txt2T                  \\ Direct copy of Counted Chars Arrays
MyT13 count type                   \\ Text?  prints a full tArray content
```

We will use eithMyT08er Str2T either Txt2T to make a copy, instead the now equivocal >COUNTED

```
MyT13  -Trailing   MyT08  Str2T    \\ Trim and Copy (be sure of sizes, no check)
MyT08  MyT13  Txt2T                \\ No need to check sizes here. Now it's safe
```

Lets check. Now using our ‘Text?’ word instead the standard COUNT TYPE (a bit lazy, maybe)

```
MyT13   Text?                      \\ Now we have a different content on MyT13
MyT08   Text?                      \\  And can copy counted arrays cleanly
```

Editing text with String Variables (a working pair) becomes trivial. As cutting strings shows:

* To virtually cut the end, just change length. A simple minus, after (addr lenght cut)
  (addr length cut -- addr length-cut) Why build a word that is a simple minus?!?
* To virtually cut the beginning we have the easy to build /STRING as what it does is
  (addr length cut -- addr+cut length-cut) Trivial to implement, and useful.

By now, we are only missing declaring String *inside* our code, as **."** does. Later.
To do that, we will need to use COMPILER to build the **S"** word, paired by the **C"** word.
This is an harder job, as the ACE manual overlooks the subject with a simple reference.
... Of to “use -1 and correct the final value later”. Not enough when we question more.

___
We use FORTH as we would use Assembler. An easier ASM, structured and interactive. Also to note:
After many mistakes with >R and R> (causing crashes) we rather use PUSH and POP (not shown here).
Even C.Moore advise it ... That shows the downside of ANSI . We all can change that (it's FORTH).
Also note we used 'I' references where R@ should be (still can be, just not here).

Strings inside words

Programs usually contains text (not feed from Keyboard). In Forth, that means “inside built words”. Text may be present on dot-print [**."**] or comment [**(**] words. Yet, ACE words being editable, their location may change. So, ACE-Forth contemplates text as disassembly exceptions.

Missing inline string words will not be regarded as such. Their content will not be shown (lost). Similar happened with float numbers, no longer compiled as Floats (there are code references showing they were). Other words compiling text inside a word will not be seen as exceptions.

String Compilers as **S**" and **C"** present a problem both similar and different than floats:
- They can be stored, but will be invisible to disassembly.
- They are Forth-83, not Forth-79. Not essential, are useful. BUT we need it to be disassembled.

We need a way to allow the use of S" to be seen. Thus editable without losses. The COMPILER word must be used with a twist of the standard. To allow disassembly, we need a compromise. (S" and C" are exceptions. But the exceptions list was pushed to ROM. How to do it?)

An editable S" word ... become a Magellan's Egg problem:
So, S" uses slightly different format, for disassembly to react correctly to an embedded string. Our inline string compilers MUST be followed by a comment (this to serve as editable container):

```
: HasStr  S" ( I'm a string )  ;   \\ Stacks the string:  Its address and length
HasStr  TYPE                       \\ So, these parameters can be used elsewhere
```

On this 'HasStr' example, the string pair is stacked as due and expected. Even more! Its inline text is visible and it's editable. Our ACE's **S"** definition has two parts:

```
HEX code (s")                       \\ The S" RunCode in ASM to build a quasi-primary
 13df , eb13 , 235e , 2356 ,        \\ as this Forth Primary should be in order
 ebd5 , d1d7 ,  d7 c, e9fd ,        \\ to speedup the RUNS> section.

0 COMPILER  S"                      \\ On compile, do nothing. The Comment compiler will.
  RUNS> (s") ;                      \\ On run, get the string parameters for the comment.
```

Question: With the parameters... can we change the string inside the word?
We can, but that's usually a bad idea. Note we may corrupt the String's borders.
Strings can be dangerous if we lose track of its limits. The PAD is safe to manage.

These words will not store a ‘)’. They allow a double quote instead.
! This is a small inconsistency. A small price... for a big reward !

And a similar C" word
We also implemented **C"** by placing the string on a known Counted cArray: the PAD.
The PAD is bit less volatile than the stack, but **C"** is maybe 5x slower than **S"**
It is also big. **S"** and **Str2t** serve all our needs... Thus, **C"** is not shown.

Also, comment rules do apply: It uses a general 16 bits counter (!) meaning an embedded string can not be a counted string. That's the reason we need to transfer it to the PAD. Why is that so? ... Unlike PAD, the COMPILER mechanism uses a 16bit counter for its space !

Why not 8 bit? It could. Whatever the reason, being uniform (16bit), code is reused. Again Why? Because the ACE needs to fit on 5k with its extras, ie , need to fit in 4K. Without constraints, an 8bits counter might be in place (no Compiler word needs 16bits).

```
: Test2 C" ( Hello, and GoodBye) ;      \\ C" word not shown, maybe later
Test2                               \\ Text should be on PAD by now, ready to...
Pad Text?                           \\ This command will show the text now on PAD
```

___
 # The coffee we are waiting... is surely cold by now.

Chapter 4 - Advanced Topics

People usually implement words needed as 'supposed', sometimes not not exactly what they do:

```
    : 2*  2  * ;                  \ --- wrong ... We want a double, not an heavy multiplication
    : 2*  DUP + ;                 \ --- right ... Here we double, is fairly quick on any CPU)
```

On the ACE, the later is 2.6x faster than the first. Even faster if DataStk is the CPU stack.

Similar happens with ROT's reverse, ROT- (Prefix abuse creates problems with other usages)

```
    : ROT-  rot rot ;             \ --- wrong on the ACE      (may guess why)
    : ROT-  swap  >r  swap  r> ;  \ --- right on the ACE      (twice  faster)
```

This 2nd way to do ROT**-** is an exception. On Forths using CPU stack, first example is faster.

ROT should be a primary, as stack re-arranger. Constantly used, CPU stack should serve DataStack.
CPU or Soft, stack management words MUST be Primaries. ALL. This is more relevant for the ACE.
Note: Immediate reason not to use ROT to build **-**ROT, is that ROT was trashed when forced to be a secondary... ROT**-** is a seldom used word it could be a 2.5x slower secondary.

A practical rule we might suggest, is:

- When a seldom used word is missing (rarely a bottleneck) optimise the secondary.
  If a bottleneck, then should be implemented as a Primitive. This may also be valid for user words as long as they are small and the user is able (or already available as CODE).

** Use * the * CODE * Word **

Forth is an ambidextrous language, where ASM is its forgotten hand. CODE word make ASM easy.
Its usage is direct, without ‘external’ tool dependencies (usually imposed by manufacturers).
It's that easy. If you have only the DEFINER demo from the manual, here's the real deal:

```
    : CODE  create  here  dup 2-  ! ;     \ TIP: precede CODE with HEX (allow it to be flexible)
```

As the FSF delivered free Tools, manufacturers imposed hidden tools and libraries...
Their products cannot include those sources without the tool maker permission.
Anyway, royalties can be demanded (maybe fair, maybe not).

Here's a CODE word example of the 2* primary previously mentioned.
Implemented as ShiftLeft(TOS)... We could also use Add(TOS,TOS)

```
    ;   CODE  2*            ; Function          ;Size   ;Clocks     \\ OPs      \ Joined
    ;     RST   18H         ; Dpop,             ;1      ;89         \\ df       \  ..
    ;     sla   E           ; SHR               ;2      ;08         \\ cb  23   \ cbdf
    ;     rl    D           ;                   ;2      ;08         \\ cb  12   \ cb23
    ;     RST   10H         ; Dpux,             ;1      ;89         \\ d7       \ d712
    ;     jp    (iY)        ; Next,             ;2      ;93         \\ fd  e9   \ e9fd
    ;                       ;------------------------------------------------------------
    ;                       ; 3.25x  DUP +      ;#8     ;=287

    HEX CODE 2*   cbdf ,  cb23 ,  d712 ,  e9fd ,  DECIMAL    \\ code is only #8 (total=15 Bytes)
```

This CODE definition for the ACE is fast to type, ASM hand-assembled, as all words we restored.
First, using the Z80 data at the ACE manual Appendix. Later with my own chart (on Book-II end).
We still hand-compile every piece of code (for whatever CPUs Z80 x86). Clocks are a reason why
We built our own tables. So we may assemble, disassemble, measure... Knowing WHAT, and WHEN.

Building an ASM routine as a Forth Primary, gives ASM the 'flexibility' of Forth.

A small DUMP utility

This tool is a programming example. One also showing bin to hex conversion.
(Notice management of stack: It is rather small. Sometimes not needed)
Sometimes we want to examine what was built. Usually data, sometimes code.

This DUMP is derived from a version shown by Brad Rodriguez. Thank him too.
It's a practical version of that DUMP tool, using another ?? quick dump tool.
Here, we'll use a pair of previously mentioned missing words: LSHIFT .. RSHIFT

Usually, 'programs' are designed Top-Down. Then developed (tested) Down-Top.
Here we see the 2nd part of the process, every bottom words also being tested.
This testing is not shown, yet done. Once done, allow focus on words using them.

```
\\ *** Low level words ***                 \ Lshift and Rshift must be loaded first
decimal     \\ just in case...
  : ><      \\ (u1 -- u2)                  \\ swap bytes of TOS (not ANSI)
      dup   8 Rshift                       \ need to load RSHIFT (its on SMALLIB)
      swap  8 Lshift  +  ;                 \ ... and yes, LSHIFT too (also there)
  : LO      \\ (c1 -- c2)                  \\ return low nybble of TOS
      15  AND  ;                           \   Notice 15=$0F (4bits masked).
  : HI      \\ (c1 -- c2)                  \\ return high nybble as LO
      4  Rshift  LO  ;                     \   Notice how Bit shift words are versatile
  : >HEX    \\ (c1 -- c2)                  \\ autonomous nybble -> hex_char
      LO  dup  10  -  0<                   \ smaller than 10 ?!?
      IF   48  +                           \ ascii '0', for '0'..'9'
      ELSE 55  +                           \ ascii 'A'-10, for 'A'..'F'
      THEN  ;

\\ *** Middle level words ***
  : .HH     \\ (byte --)                   \\ print 1 byte   (2 hex digits)
      dup  HI  >HEX  EMIT  LO  >HEX  EMIT  ;
  : .AAAA   \\ (addr --)                   \\ print unsigned (4 hex digits)
      dup  ><  .HH  .HH  SPACE  ;          \ and a space = 5 chars

\\ *** High level words ***
  : .Byte   \\ (a -- a+1)                  \\ fetch & print byte (advancing)
      DUP C@ .HH SPACE 1+  ;               \   = 3 chars
  : .Char   \\ (a -- a+1)                  \\ fetch & print char (advancing)
      DUP C@ space
      dup 13 = if drop 32 then
      emit SPACE 1+  ;                     \   third char  (synchronised with .Byte)
  : .Dump   \\ (addr -- addr+8)            \\ print 1 line of 8 bytes (advancing)
      dup                                  \ will use addr twice
      8 0 do .B loop  drop                 \ use addr copy, discard result
      8 0 do .C loop  ;                    \ use orig addr, leave addr+8

\\ *** Final Tool words ***
  : ??      \\ (addr n -- addr')           \\ Print a Dump of n lines, 8 bytes each
            \\ n lines dump, after showing Hex Addr. Easily repeatable [ 'n' ?? ]
      swap dup .AAAA CR
      swap 0 do .Dump loop ;               \ Usage Example: 15360  3 ??  5 ??  8 ??
  : DUMP    \\ (addr -- addr+64)           \\ Very short, easy to edit (aka configure)
      4     \\ Dump of 32 bytes (4 lines) after Hex Addr ... 4 lines is fine. But
      ??    \\ we can easily reconfigure the number of lines: Just Edit this small word
      ;     \\ DUMP User_Area example:   FIND FORTH   DUMP  DUMP  DUMP

\\  Usage example:                         \ Some SMALLIB elements must be loaded first
\\      FIND FORTH   DUMP                  \ Here examining a Voc structure
```

4.1 Lesser Work, Better Work

A small introduction:
When I got my first (and only) programmable calculator, a Texas Instruments, I've exercised its usage by doing a search (by approximation) of the best smaller numbers fraction of Pi.

After some 20 hours, the definitive winner (best close fraction) was 355/113 found after ten hours. That pair gave a precision not surpassed by any other in the 3 digits class. Beside that, it a beautiful and easy to remember pair: 11 3 ... 3 55

Decades later, I've found a reference to those exact numbers, made by Chuck Moore (pleasant moment). Point is: Moore also mentioned scaling as a fast calculating method for casual decimals (Moore fast) as multiplying constants by 100 or 1000. By 128 or 1024, is useful for variables (=Fixed precision). ... As would be done in Assembler (when to assemble was close to ingenuity).

Avoid FLOATS when without a Co-Processor
To a real programmer, Reals (Floats) are not 'needed'. He knows integer arithmetic to be faster. Also knows Floats to be slow without a Co-processor, and extremely slow on 8-bit CPUs. BUT:

Having resident FLOATS was one of the ACE distinctive marks. Floats added conveniences to a FORTH system (not bought separately). As FORTH was an High Level Language and potential users demanded FLOATS. These were respectable (made sense, before Rom space was found reduced to 5K).

Forth was the only language allowing new tools to be developed internally instead externally. Other would develop tools and programs, usually compiled on a Mini-Computer under Time-sharing. Only then loaded into the target Micro. Micros only option for many years, was a batch of general routines (BASIC) with the chance to load those commercial programs compiled elsewhere.

Forth on Rom changed all that! On a cheap computer, made it a game changer, scaring the status-quo. But only IFF the hardware complied with the goals. (Floats only needed to avoid 'equivocations') Therefore all sorts of adjectives projected AGAINST, exorcising a more educated enthusiasm.

ACE-Forth Floats used 4 bytes: 1 byte for Exponent, and 3 bytes for Mantissa.
As such, an FP value was similar to a DOUBLE: Easy to manage on the stack.
Code was condensed enough (647 bytes) to be included as a bonus.
That included compiling Float Literals (later removed).

Again, FP was demanded, considered essential for any SERIOUS work.
Were needed? Most real programmers knew a bit better, dealing with calculus in other ways (as already described). Other reasons for being 'requested':
- It is EASIER and FASTER to program with FP though the result was slower.
- More important, it was 'expected', as for trigonometry, people unaware of alternatives.
  What was appropriate? There were many factors, Mini-Computers being the beacon to follow.

An opposite example: FP was a bit too much for 8 bits. Even so, there were other reasons::
A friend, a civil-engineer, was delighted to calculate buildings structures) on his home 8bit, instead to pay and wait for the results from the Lab that (by the 70's and 80's) delivered that service (processing faster than the traditional ways, by hand with a slide-rule and printed tables.

It didn't mater how slow Floats could be: Using 8 bits was faster than waiting for the service.
Note that using Integers is the equivalent to use a slide-rule. Just not by hand.
But programming that way demands some care, even an extra care.So, was FP abusive on 8 bits?
Floating Point on 8bits WAS useful... when time was not a problem.

The question remains: Did Floats made sense on ACE-Forth? Yes, and No.
So many things stopped to be what they would...There is no B/W answer to that.
Nor are valid any linear reasons to argue (or quick read)... threaded on their webs.

When routines are words

A problem with lack of ROM space (due a flawed non-design) is faced when needed routines were forced not to be shared. I.E, forced to reside as single words. What do we mean?

When building words, we can build them as Secondaries or as Primaries.
Secondaries use Any kind of Forth Word. But Primaries run as ASM, routines called in ASM will end with an ASM 'RET'... Without ROM space, a called once routine becomes a word.

On Primaries we may need to call such routines turned Forth words.
We may also need to run operating Forth words already available, single or sequences.
Whatever the reason, we may need to Go-Forth from ASM.

Only on ROM we may be allowed to do the reverse: To Go-ASM (for whatever reason). Why only on ROM? Because Hybrid words cannot be disassembled. We may also CALL a Secondary, but only if that secondary resides on ROM (then with a fixed address).

Transitions allow the creation of Hybrid words

Go-ASM from Forth (and eventually to come back to FORTH) is not legal on the ACE as it breaks its disassembly scheme: All code being Forth words (only ROM words are exempt).
It's a question of what dispatcher is in control. A change must be polite.

To Go-Forth (as to Go ASM) depends from the implementation.
On the ACE where IP is saved on the the CPU stack, it's trivial: A primary will Go-Forth with:

```
    CALL (a Forth-Word) \\ will Enter Forth, will continue on Forth.
```

While to enter Code, as long as only relative Jmps are made and are no Hard coded addresses, an hidden primary will be called, just a RET... Very convenient.

Transitions save code space, are easy to implement

A Simple example, of Entering Forth to use its Words-Set, ending as a Forth Secondary:

```
    <preceding ASM code>
    CALL NEXT                \\ Enter FORTH
    <Forth sequence>         \\ Cannot be edited
    <semicolon>              \\ This exits the Primitive
```

Another example of the same, here going Forth, and then back to Code:

```
    <preceding ASM code>
    CALL NEXT                \\ Enter FORTH
    CALL <First >            \\ Cannot be edited
    <following Forth sequence>  \\ Cannot be edited
    <Hidden RET word>
    <continuing ASM code>
    JP NEXT                  \\ the usual JP iY
```

Had D-Stk been the CPU stack, transitions would be slower, while the whole would be faster.
Such balance, this an many other choices, are details that define an implementation.
These also show insight and mastery. ACE-Forth use such transitions.
(Everything shows mastery, except the initial architecture)

___

We have used these methods on a few of our private/undisclosed Words.

4.2 Scaling, and Fractions

SCALING Integers in place of Reals:
Instead Pi=3.14 we can use Pi*100=314 ... instead demanding decimals because are usual.
FIXED precision uses integers, it's fast. We decide how precise it need to be (if we can).

To understand WHEN to use fixed, an example we may consider is Time:
- Minutes, seconds are not decimals... We convert Time to seconds (or milliseconds).
Likewise, we can convert a Real. So the decimal part is included (arbitrary scaled examples):

We can convert a value to its decimals by multiplying it by 100, or 1000 (or any other value):

```
    Fixed Scaling with Integers:    ~1234.12    ->  123412.      (scale is 100)
    Fixed Scaling with Doubles :   ~12345.1234  ->  123451234    (scale can now be 10000)
```

Closer to the hardware (faster) we can use 128 or 1024. Then work with that temporary integer.

```
    Fixed Point with Integers (16bits=9+7)   ->    9bits:7bits   (scale is 128)
    Fixed Point with Doubles  (32bits=22+10) ->   22bits:10bits  (scale is 1024)
```

We use scales all the time: mili or micro, K or Mega. An Arithm Scale (constant) gives us Integers.
Are used for thousands of years, everyday without notice, to avoid decimals, avoid mix units.
To multiply or divide, a few corrections may be ‘convenient’. So to avoid excessive sizes.
Thus, we want to de-convert after a multiplication, and re-convert after a division.

Or we may use a log scale (on constants). It's faster: Just add logs for a mult, slide-ruler like.
Similar (not the same) was to the use of dozens (instead tens) in traditional commerce.
12 is easy to factor, easy to divide by 2,3,4,(5),6 ...(5)? Not really, but it's *10/2.

Common mistakes on Scaling:
In the age of easiness, blind training and daily Fluoride, old wisdom goes forgotten.
Sight demands openness! We can scale quantities as long as we see what we are doing:
Consider a worker raises a wall, 3.14 meters/day. For 3 workers that equates as...
As 3 workers x 314 cm/day, not as 300 workers 314 cm/day. Now try with areas. Ha!

Multiplication adds members scaling , remind to scale down. Division subtracts their scaling.
It's even faster with SHIFT when using scales of 128 or 1024. While simple, its can be tricky.
It is engineering, so effective it's done for thousands of years: Hand work demanded ingenuity.
Regardless of how fast a computer is, it PAYS to keep old wisdom known! (It teaches about us too)

FRACTIONS can be even faster:
Above, we suggested Pi as 314... But the fraction pair 355/113 is 3.14159292 MUCH more precise.and avoiding accumulation of losses The ubiquitous */MOD word is perfect for those pairs. Example:

```
        : Pi 355 113 ;
    To multiply n by Pi ..  Result to be n*(355/113)        \\ We may use a scaled ‘n’
        n 355 113 */MOD     Becomes     n Pi */MOD          \\ ... and get a scaled result
    To  divide n, by Pi     Result to be n*(113/355)        \\ Same here, to get it scaled
        n 113 355 */MOD     Becomes     n Pi swap */MOD     \\ ... on our chosen Fixed-Point
```

To be practical, we may define *Pi and /Pi functions... when these operations are much used.

You need no decimals if you find your fractions. If not, 123.45 is always 12345/100
I.E. not wanting fractions beyond Pi (and other constants) we can use (FIXED) scaling.
In both cases, we need to do it carefully, if only occasionally. Also need to test results.

After a needed extra care, it flies. How fast? There's no exact value. As any Word, it's done!
(It can be 10..50x faster than Floats, depending from the programmer's choices and solutions)
Do remind: Even a small factor of 10 is huge, making 4MHz to look 40MHz...a decade ahead(!)

Fractions are solid engineering. FLOATs are a convenience, but slow without a Coprocessor
- CHOOSE YOUR SCALE... It MAY BE all you need, faster!
- KNOW YOUR FRACTIONS They will keep their precision!

A fast SIN hack

Sinuses are a common part of programming. Usually done trough tables and interpolation.
Such tables would be on the remaining 1K (of the absent 3K BIOS, for a total of 4K).

In the resulting absence (of those tables) we need either to load one (table),
either to calculate sinuses. We can use the series method (with a few factorials on
a small built table). No matter how fast (using factorial tables) it's slow due divisions.

No tables, no joy ?!?... Lets rewind a dozen centuries, away from this only 5K available ROM:
Small 8bit CPUs are too slow otherwise, while precision is just a matter of using doubles,
or floats. Who would do that? I've seen serious work done, taking several days non-stop.
Later done in seconds on 32 bits... and a co-processor. Lets go 'retro' for a moment.

The *Bhaskara a* SIN approximation, a fair approximation by an 7th century Indian Mathematician,
can be adapted to using Integers (UNSIGNED). We have use 256 scaling to follow usual accuracy.
It's faster than decimals, and shorter. Precision is better than 1%, in Steps of half degree.

It's as fast as consulting a table and interpolating its data.
Input : 0.. 180° (2 quadrants = 'sky arch') NO negative results allowed.
Output: integer, usually the lower byte (except on 1.00, not a fraction, then = $0100).

Because it's fairly precise, it could accept halfs, even quarters of degree (tested OK)
After evaluating 8 different options, we chose to limit usage to halfs of degree.
This keeps usage simple: An extra parameter 0|1 for half degree.
Input range goes from 0..90° (90..180° also accepted).

We do not want to use Floats. Nor use a 100 factor (less precise, slightly slower)
Nor to force a source of confusion if an angle was expressed in halfs instead of degrees.

Fair Warning, as rule of Thumb
This is an 8 bits CPU. Nor a Math engine with tables available. Lets get real:
Accuracy is 1/256 50% of the time, 1/128 40% of the time, 1/64 bellow 10°.
    For Physics use something else, maybe a 32bits PC with 4MB (an overkill).
    We have chosen to deliver half degrees, and a scale factor of 256.
Output is a byte (0..256) representing (0..1)*256.

USAGE:
Reasonably accurate (>99.5) 50% of the time, it allows half degrees on input.
SIN uses 2 parameters: degree and any 1/2 of degree

```
    Bsin(30,0) degrees is called with ...
        30  0  SIN      \ Result is 128 (remind factor is 256)
        Note the second parameter is for half degrees
    Bsin(30,5) degrees is called with ...
        30  1  SIN      \ Result is 130 (as factor is 256, SIN(30.5)=130/256
```

As the RND routine, from the 90's, this one was not (?) available in 1982.
Difference was a 13 centuries gap... though in the opposite direction. And speed?
Forth-only implementation SIN took 14ms. As Primary, we expect ~7ms (still need */MOD)

After the above exercise:
As we accepted <0.5% error and single degree steps, we may use a factored (*256) SIN bytes table.
Delivering SIN(degree)*256, can be reduced to 90 bytes (1°). Or to 48 bytes, for plotting.on 2x24.
    The result satisfies both gaming low precision, reduced memory and extra speed.
    Forget not that 256 does not fit on a Byte (0..255)... Hack your solution.
Using a table, speed is instantaneous... C.Moore advises not to calculate constants
Having reduced an infinity to a small set of 90 (maybe to 48 or less) that advice is applied.

___
Some time lost, examining needs and chances, pays on results.

4.3 Expanding Our System

Before we start, Bootlenecks deserve a word or two
 FORTH is an Ambidextrous Language, usually limited to left-hand usage.
 Its'Left hand being Forth Words... 'right hand being faster Assembler.
 The ACE manual (Chp 25: MACHINE CODE) gives all indications one needs.

 Application bottlenecks, as essential actions, deserve special attention.
 First, refining the method used. Later, CODE-ing its critical portions.
 Common sense dictates steps to be applied. (scarce after "training")

 First, we should define and adapt the algorithm.

Words needing optimization can be identified by

1. frequency of usage (the usual on any language)
2. their Secondaries Level (number of Nest-unNest, aka Enter-Exit)
3. amount of arguments passed (only valid for soft-stack implementations)

```
Tree levels suggest accumulated Enters, Exits, Args passing. How to calculate them?:
    OVER was implemented using >R  DUP R> SWAP  (all primaries, all Snumber=0)
    As a secondary, we add 1 to the bigger S-Level on words used (was 0)

    Divisor (/) uses /MOD SWAP DROP (we will ignore the primitives)
    So we'll add /MOD Secondary Number (We do not know it yet)
      /MOD uses U/MOD ABS and several primaries
        U/MOD uses UM/MOD (hidden), then uses ABS
          ABS uses ?ABS (an hidden word, usually a primaries =0)
  *** Thus, on this callers chain: ***
          ABS has a S-Number of 1 ...
        U/MOD has an S-Number of 2 ...
      /MOD has a number of Three ...
    Divisor (/) has a number of 4.  (There's waste here!)
```

This reflects a number of NEST-UNNEST into words. It's useful, not so much because NEST-UNEST (which in the ACE is fast), but suggesting inefficiencies placed on Secondaries accesses.
BTW: If data/args Stack is the CPU Stack, each delay is on NEST-UNNEST... but single!

No matter the architecture, Secondary levels are useful as NEST-UNNEST (also as Size Tip).
On the case of the ACE, not as useful, yet we have used it to find hidden bottleneck words.
Such seemed to be the case of the very curious ?ABS (not standard, obviously made on hurry).
?Negate was hidden, ?ABS replaced IfN0Negate (name was kept) used by ABS, and on Arithmetic.

A fun curiosity (it's advisable to have one):
To note Secondary Numbers are similar to Erdos Number. (His own joke, sarcasm would be 'mirabilis')
That joke moved a few decades ago into the "*Small World, with 6 degrees of Separation*" (an average).
We mentioned 'similar', because of a difference: We account the longest path, not the shortest one.

Naturally, the joke need fixes. To be useful as much as usually enlightening.
Just for fun: Generalising the notion (anyone can use this), we may attribute 0 to self,
1 to close family, 1+ to every indirect connection.following. (We got curious personal results).

Similarly to the Tree-level number, of Secondary words, such correction would not be enough:
Quality is a dimension to add. So we could/should then add 0 when ‘close’, 1+ for 'casual'.
Other dimensions can be added, as long as kept short ... Again, with curious results.
This also shows that even though Erdos Number was a private joke, it can be useful.
Great fun!

 # The world is too small... We all need to smile a bit more.

The case of CASE

On<u>**CASE**</u> ... A fast CASE.implementation, usable with or without out a safety net.

There are many ways to implement CASE. This one is for linear cases (using a vector array).
Easy vectors, implemented as Secondaries, are ACE friendly in the sense of allowing content
to be relocatable, editable. And faster than several ELSEIFs the CASE construct only mimics.

This solution implements <u>editable</u> vectors.(such vectors have many other uses).
Our OnCase word acts on available entries after it (editable because in a word) into
a vector the ACE can disassemble (list & edit). Yet, it demands a correct index value,
as usual on arrays. Goal achieved, the downside is that references must be correct (...).

A Study implementation (shared a decade ago). Short and Much faster than CASE..ENDCASE

```
   :  OnCase  ( n --)      \ n is [1 to n], user friendly
       1-                  \ correct n to zero-based
       dup +               \ N*2 is ListOffset
       R> +                \ caller=List, plus Offset=Action call addr
       @ >R                \ Get action addr, let semicolon execute it
   ;                       \ caller was dropped (R>) and replaced by the chosen action (>R)
```

In spite serving the purpose, and in the Moore's Forth spirit, was overlooked due the usual
not “what we want”. So many times opposite to "what we need", or even close to “make my day”.

<u>We built its successor as a COMPILER structure, and its Primary</u>
Table Vectors, as the OnCase(n) type, call a single full procedure.
This way, a Vector Table can be Edited and redefined. An editable Vector.
The later VECTOR> directive allows post-processing, as much as pre-processing.
It does not Check Indexes. Not his job, that can be done before its invocation.

<u>**Example of usage**</u> Also to test the utility word (when refining it):
Define a few example procedures. A few visible ones (messages)

```
   : MSG1 ." one/" ;       : MSG2 ." two/" ;      : MSG3 ." three/" ;
   : MSGx ." ErrOut/" ;    : MSGy ." Oops!/" ;
```

Now we are ready for a usage example

```
  : TestVector ( n -- )    \ Be Sure of n... or crash
   \ Option: Before OnCase anything will run, as correcting n to one based
       10 -                \ VBegin correction. Range 11 to 13 (actions 1 to 3)
   \ Option: The following testing may suggest usage to be hard. It's a caution
       dup 3 >             \ This example has 3 actions, note this test is optional,
       IF MSGy EXIT THEN   \ serving to avoid an out of range error (xy;). Or a crash
   \ The above is not imposed, maybe not needed. A Temporary debug-check, may be.
   \ Bellow, the real editable vector (sequencial CASE, 1-based, 3 actions)
   OnCase                  \ After, only one action runs. Then exits!
     MSG1                  \ These are example procedures
     MSG2                  \ same
     MSG3                  \ same
   \ Option: One or two messages, as an alternative debug measure
   MSGx                    \ x intended message, to serve as a crude safety warning
   MSGy                    \ y not intended, exemplifies absence of the range test above
   ;                       \ ; If value point here, no message. After here, it will crash
```

This is not exactly entry level. But it is simple enough and covers several options.
It includes no safety checks, maybe needed for public usage (added on the test example).
An alternative is to use COMPILER (much better) and/or a much faster CODE word (or both).
For efficiency, we would always use 0..n-1 ranges. For readability, 1..n.
Our example <u>starts with "11 TestVector"</u> up to 14, wrong values not checked.
A check MAY be advisable, but later. The example shows it, but as a choice.
Needed? A assert word can be inserted, for debugging (forget not to remove).

Just Vectors

A solution for Vectors
Vectors are the Achilles kneel of disassembly, here solved with a new COMPILER word.
COMPILER allowed the OnCase snippet to evolve, into more usable Vectors (see below).
Developing clarified some misconceptions on this tool built to ease programming.
(After the previously solution), the new COMPILER and CODE words are a teaser.

Away from fixed presumptions
<u>It is not an easy task</u> when not familiar with COMPILER use (as we were not).
It's beyond the ACE manual description. While quite informative, the details without needed
feedback as confimation are quickly overlooked. This particular control word was hard to build.
(its fair to confess) than others more 'regular' in face of inexperience (even having its sources).

On COMPILER we faced blind spots. Regarding Vectors, we also had unsolved questions:
 * What would <u>be most effective</u> mechanism to build ? (defining a chain of actions)
 * What would <u>make more sense</u> for a clean usage? (a goal achieved, made practical)

Goals reached!
We are pleased with the solution (and its implementation): Versatility, simplicity, speed.
Usability is well expressed on the changes shown to the already presented TestVector word
The example is now renamed, being slightly different. It seems now so simple, a good sign:

```
=== USAGE ===
    : TestCase ( n -- )     \ ! Be Sure of 'n' !... Executing bogus addresses with crash
        ( Vbegin  - )       \ Pre-Processing Range adaptation = Vbegin -> 0   (usually 1- )
                            \ Because Vectors start at 0 (FORTH favours zero_based)
        VECTOR>             \ Begin-Block for the Vectors linear Array... Editable
            ERRMSG          \ Wasting zero slot to an exception may remove the adapt stage
            MSG1            \ ...allowing to organise actions as a more readable 1_based
            MSG2            \ Identifiable Word actions replace Blocks: Cleaner!
            MSG3            \ A faster solution.for an old problem: Forth's best!
        <VECTOR             \ End-Block tests Begin-Block presence. Then initializes it.
                            \ The above is a Vector container.It returns a slot address
        @EXECUTE            \ Post-Processing Behaviours are added to the Slot address
    ;                       \ Here, Execute content. Sometimes nothing (config Vector)
```

COMPILER allows to expand the language: With new Structural-Words, or Utility-Directives.

```
=== Testing ===
    2 TestCase  --> Two/ OK     \ As range is 1..3
    4 TestCase  --> <crash>     \ If your code allows invalid indexes, something is wrong
    0 TestCase  --> <crash>     \ While testing, consider to filter requests
```

We implemented it as an inline Array (addresses are still found at runtime).
This shows that programming Vectors can be done on ACE-FORTH! Cleanly, safely.
When <u>CASE use</u> is linear. it needs not always needs a slow succession of IF-THENs.
<u>It can be fast</u> ! It measured an access speed of 1.6k vectors/sec (no @execs included).

We insist on the following practical details, though small:
For efficiency, we would always use 0..n-1 ranges. For readability, 1..n.
But also: Checks are not part of essential words, not once final usage is correct.

Final user is not responsible. The programmer, is!
Thus, fool-proof libraries do NOT belong to Forth methodology.
Forth respects the programmer, its assumed to be competent and responsible.

<u>In short</u>: Responsible Cooperation allows efficiency. Respect is also needed.
Finally: Efficiency is not just speed. Ease of use and integration also counts.

An HASH away

In theory, CRCs offer a perfect distribution. In practice, data is decisive.
Quick Hashing can be very useful. Options go from crude CheckSum to a Quick Hash.

    All these are 16bits. Better fetch 16 bits values in the loop.
    If size is odd, last byte in the loop should be read as (byte,nil) or (byte*256).
    Once more, Forth Do-Loop (excluding limit value) helps us optimising such construction
    (on small words)... Using size as limit, the loop ends on (n-1). Warning: Check Size=0.

PSEUDO-CODE:

```
1 - Checksum (single OP)           *** CHCK *** (RelativeTime=1)
    Not to be used, unless for lesser needs.

    Init:                          \\ Optional, better than zero
        Hash  := $352b             \\ iff 8bits use $35 as Salt
    Loop:
        Hash  := Data + Hash       \\ worse case scenario would be zeros

2 - Positional CheckSum            *** PCHK *** (RelativeTime =2)
    Crude transpositions detection, faster than ‘counted’ CheckSum

    Init:                          \\ Optional, better than zero
        Hash  := $352b             \\ Iff 8bits, use $35 as Salt
    Loop:
        Hash  := Hash SHL 1        \\ worse case scenario would be zeros
        Hash  := Hash + Data       \\ ... but now only if 16 or more

3 - Triple Spread                  *** HASH *** (RelativeTime =3)
    Simple, as above. But with a better Spread. Very fast. NOT CRC!
    Our private method, it uses 3 different OP ‘spread’ properties

    Init:                          \\ Optional, better than zero
        Hash  := NOT($352b)        \\ iff 8bits use Not($35) as Salt
    Loop:
        Hash  := Hash SHL 1        \\ worse case scenario would be zeros
        Hash  := Hash xor Data     \\ Xor and ADD are great scramblers
        Hash  := Hash + $352b      \\ Scramble zeros. Iff 8bits, use $35

4 - Shift and Fold (carry on Tips)  *** SASH *** (RelativeTime =8)
    Essentially the previous, here we simulate a Rotation for Shifts-only CPUs... (Why?)
    Rotation has a problem : WHAT BIT will be Carry (or receive it)?. With no fixed
    target but many, most CPU above 8 bits offer no bitwise rotation, only shifts.

    Init:                          \\ Optional, better than zero
        Hash  := NOT($352b)        \\ iff 8bits use Not($35) as Salt
    Loop:
        Tip   := Hash AND $C000    \\ Rotating twice. Iff 8bits, use $c0
        Hash  := Hash SHL 2        \\ 2* 2*
        Hash  := Hash XOR Data     \\  xor
        Tip   := Tip SHR 14        \\ Tip contains 'Carries', iff 8bits use SHR 6
        Hash  := Hash OR Tip       \\ Place 'carries' on Tip into their new position
        Hash  := Hash + $352b      \\ Scramble zeros. Iff 8bits, use $35
```

Implementation of any simple non-CRC hash's:
We can either Hash a memory Block, with (address, size -- Hash).
Or a direct single value, in a bit slower way, with (Hash, Data --- Hash').
This being an excellent example of choices: Arguments, Implementation and Naming.

Nice Hash

Implementation of Triple Spread Hash:
We can either Hash a memory Block, with (address, size -- Hash).
or in a bit slower more Universal reduced way, with (Hash, Data --- Hash').
This being a wonderfull example of choices: Arguments, Implementation and Naming.

We may call the memory block Hash as 'cHASH', reminiscence of cMOVE's (aStart aEnd size ---).
Or use a more efficient full word Hash (considering a NIL for extra chars) as result is 16bit.
It's more Forthish too. We'll keep the previous HashValue as 2nd parameter (result), with data conveniently on Top-of-Stack (to be processed).... HASH is ready to be a fast CODE primary word.

First built as a Secondary Forth (always a good practice, when testing) needs seldom needed bit-Rotation words (these are a good exercise). Rotating Left and Right, 8 and (here) 16bits.
We'll want to keep Carry with the same care low level division keeps 'remainder' available.
Will use the same proven strategies on our bRot routines -> (carry, value -- carry, value).

Simulating bit SHIFTs and ROTATIONS
Notice than usage of our bRot will demand 2 operations: bit-shift, and later 'carry' insertion.
This way our 8bit bRot will be Universal, allowing any other size operations (16, 24, 32, 40, etc)
These low level OPs building do not represent usual programming but system expansion. Enforcing two steps we enforce, we will will not call it bRot. We will call SLIDE to the first step.
As 2* is easier to replace, we chose LeftRotation. (Bits Right, named Slide**-** is not shown.

```
: cSlide ( carry', value'  --- carry", value" )  \ 8 bits Word-OP for LEFT bit Rotation
    2* +              \ Shift left plus carry (use " dup + " if not using the Library Primary)
    dup 256 and       \ ... Bit8...
    8 RSHIFT          \ ... Go bit0
    swap 255 and ;    \ keep args order, enforce 8bits

: Slide ( carry', value'  --- carry", value" )  \ left bit Rotations... A 16 bits "RL reg"
    dup 8 RSHIFT >R   \ Isolate HI byte and save it
    cSlide            \ change LO byte, carry actualized for next
    R> swap >R        \ replace with High (now 8bits)
    cSlide            \ change High, carry actualized for next
    8 LSHIFT          \ Restore High as HI (16 bits again)
    R> +  ;           \ join HI+LO bytes (carry bellow, untouched)
```

Slides are atomic operations, preceding both Shift and Rotation and allowing other bit sizes.
Rotation needs to place Carry (as following Operation). On left rotation that's a simple '**+**'

Now that we have the needed tools, we may translate the following Pseudo-Code into Forth

```
    Hash  := Hash SHL 1         \\ worse case scenario would be zeros
    Hash  := Hash xor Data      \\ Xor and ADD are great scramblers
    Hash  := Hash + $352b       \\ Scramble zeros. Iff 8bits, use $35
```

As our tools were tested, shown correct, next step is now straightforward.

```
: HASH ( Hash, Data --- Hash")  \\ 16 bits (cHash not needed)
        >R            \ Save Data
        0 swap SLIDE +  \ Rotate Hash = Add Slide Value to the resulting carry
        R>  xor       \ xor Hash' with Data, then
        13611 +  ;    \ add Disperser (as previously designed)
```

In the process, we found errors (not shown, corrected). Errors are common, but words are small.
Each new word was tested after previous is correct and usable. Later may build them with CODE.
! Primaries will be faster than usual: The Z80 has dedicated Rotate bit OPs.

<u>Exercise</u>: Build the cSlide**-** and Slide- right rotations. (Peeking is rarely enough.)
<u>ASM Note</u>: Testing is easier on Forth than on ASM. (CMOVE is an exception, the Z80 has it.)

Use Atoms

Triple Spread Hash ... goes CODE
Forth stimulates code reuse. CODE words being those small, usable pieces make Forth.

CODE words MUST be independent, not 'fixed' location dependent. That's very efficient.
SHOULD be simple operations to replace or extend CPU OPs, serving Secondary words.
This enforces simplicity. Words short and reusable for the programmer to glue.

To build HASH, we are tempted to copy the ASM bRot code into a HASH primary... Wait!
By Forth philosophy we do the reverse: Build CODE bROT words, a COLON Hash using them.
What if HASH is an application bottleneck? We are free to rebuild it as CODE if needed.

We know we need Overflow Management: CbRot (carry', value' --- carry", value")

<u>cSLIDE in Asm:</u>

```
    D-pop (BC)          \ value'                        \\ cd 4e 08     \\ 4ecd
    D-pop (DE)          \ carry'                        \\ df           \\ df08
    ld  B,0             \ enforce Byte                  \\ 16 00        \\ 0006
    ld  A,E             \ to extract Carry'             \\ 7b           \\ 1f7b
    RRA                 \ Carry' to Cy flag             \\ 1f
    RL  C               \ moving bits, and Cy           \\ cb 11        \\ 11cb
    ld  E,0             \ Reset E for Carry"            \\ 1e 00        \\ 001e
    RL  E               \ Cy flag into carry"           \\ cb 13        \\ 13cb
    D-push (DE)         \ carry"                        \\ d7           \\ 50d7
    ld DE, BC           \ Not an OP... it's a macro     \\ 50 59        \\ d759
    D-push (DE)         \ value"                        \\ d7
    JP (iY)             \ NEXT                          \\ fd e9        \\ e9fd
```

This was simple. (Though we forgot both "enforce byte" and "keep Carry as bit0-only")
Now it's time for SLIDE, so to avoid double work and its (heavy) Arguments Operations.
Actually, once we only needed SLIDE, we ignored cSLIDE for now... Here's why:

<u>SLIDE in Asm:</u>

```
    D-pop (BC)          \ value'                        \\ cd 4e 08     \\ 4ecd
    D-pop (DE)          \ carry'                        \\ df           \\ df08
    ld  A,E             \ to extract Carry              \\ 7b           \\ 1f7b
    RRA                 \ Carry on Cy flag              \\ 1f
    RL  C               \ moving bits, and Cy           \\ cb 11        \\ 11cb
    RL  B               \ same                          \\ cb 10        \\ 10cb
    ld  E,0             \ Reset E for Carry"            \\ 1e 00        \\ 001e
    RL  E               \ Cy flag into carry"           \\ cb 13        \\ 13cb
    D-push (DE)         \ carry"                        \\ d7           \\ 50d7
    ld DE, BC           \ Not an OP... it's a macro     \\ 50 59        \\ d759
    D-push (DE)         \ value"                        \\ d7
    JP (iY)             \ NEXT                          \\ fd e9        \\ e9fd
```

HEX CODE Slide (carry', value' --- carry", value")

```
    4ecd , df08 , 1f7b , 11cb , 10cb ,
    001e , 13cb , 50d7 , d759 , e9fd ,
```

All we need is Slide... Slide is all we need :
SLIDE double work was our goal. (Eventually, we may want the reversed, SLIDE-)
Both are to keep... SLIDE transmits to our HASH word the benefits of a CODE.word.
Achieving our goal, we got an essential operator. (A good example, was it not?)

<u>Forth advises Atomic Primaries, not Molecular Primaries</u> ... Bricks, not Statues.

___
 # Short Primitives can be built by hand. (All Primitives we shared are hand-made.)

Hints on Packs

'Packs' of data, joined under a name, are called 'Records' on Pascal ('structures' on C).
Unlike an array, these can be of different types. Each field having its own data type.
Example of a pack: A ‘Contact’ record [Name, Phone1, Phone2, E-Mail, BirthDate]

Inside of a Pack:
 * Head: name_word *(Note: Packs are the base on static Objects. Later, of dynamic ones)*
 * Body: CFA, index_field[first_field_pos .. last_field_pos] (Index is a Rel-Locs Vector)
 * Data: [first_field_var .. (last_field_var)] (This is a fixed Body extension

To simplify, also for efficiency, fields are usually of constant length. We can do better.
It's also usual, on high-level languages, to allow alternate fields. Packs as we designed, allow different lengths and ‘alternate’ fields. Are independent, self-managed Forth Types.

By Forth's philosophy (pre-OOP) every Type (ClassInstance) takes care of itself (its CFA job):
- Each entity delivers a needed address. Also reason why Packs are more versatile than usual.
- PRE-oop due single behaviour. Procedures using location given may be ‘hidden’ on a VOC.

Behaviour: \\ Similar to onCase... A start for Packs, a fair hint
On the Packs we built, we allow a field to be of ANY Type... Even a full Body (no header).
That will orientate our design and CFA code (the DOES> part). Notice VOCs are Packs already.
So, this an exercise on dynamic easy changes. Were removal can be a replacement by a NOP/NIL.

!!! A pack may be relocated: Indexes are Relative Positions, NOT Addresses !!!
To satisfy our goals, a Pack should be an index to fields (ie, relative positions
We could define ShortPacks with byte Indexes. Here we use 16bits with code similar to:

```
    : Array.Index  swap 2* + ;  \\ But this is not a single step array, the sequence.is reversed.
```

Start in not on TOS and above our requested index. What we want is different. Let's nor overwork.

```
    : P.ITEM 2* @ ;             \\ In the absence of 2* primitive, replace it with (dup +)
```

The dot is a mere char convenience. And “ITEM” because we are extracting it from an Index.
We built an Index with the intention is to reach any Type of field: It can be a Pack too!

To allow shared fields we can use an extra index slot to access the shared location
Fields can have its own CFA, be an Obj instance. Problem is, User Types may change addr.
(The hard part, being in the blind, was to clarify the WHAT ... The HOW is usually easier)

Usage:
1) Usage is simpler than the above strategy described. <u>Packs are used as an Array would</u>:

```
      3 MyPak     \ runs CFA (index, Pak --- FieldAddrExec) ... thus executing Field 3
```

   Example: An integer Var -> Var address (it could be a body -> CFA to @execute)

2) To use more complex Packs, as when get getting addresses, we similarly use an index body accessible by preceding a Pack with a zero. (operative analogy: a directory tree)

```
      0  MyPak    \ Self ... Field 0 execution leaves MyPak Index-Address on stack (--- PIdx)
      3  P.Item   \ Field 3 consultation... (Pack, Index --- ItemAddress)  Can be another Pack
```

   This ready to @EXECUTE when pointed content is [Type | Content]. (should be a fixed addr CFA)

To exemplify different actions, we could also execute the field directly with:

```
      0  Mypak    \ Self ... We may change context to allow “operators over-loading” as in:
      3  P.exec   \ Field 3 execution (PkAddr, Index ---ItemRun)  Different packs, same names.
```

   *T*o make this work safely with user types, CFAs should be pointed by a table (a Voc could do).

VarPacks and TypePaks are our ticket to 'Records'. Also to static ‘Objects’ (yet manual).
Question is: Do we really know what we want? Beyond those usual languages fixed options?

In other words: Are we comfortable exploring a freedom forest? Or will cross it on tracks?
Notice these are completely different travels: To discover by foot, or to reach by ticket.
Versatility can be daunting, when ‘education’ is limited to repetition trained... QED.

The TO conundrum

Some uses of Konstants are more practical, though slower, when defined as VALUE or cVALUE.
Make sense, when Variables rarely change. The usage of VALUEs makes code more legible.
There is another advantage: The defined value manages itself (reason to be slow).

The original TO used CONSTANTs. Thus faster use but slow modification.
Notice that constants do not leave an address, the modifier must be postfix.
Because postfix, on the ACE it causes a disassembly problem, quickly solved with:

```
    0 compiler >K  RUNS> @ 2+ ! ;
```

This only works inside a defined word (colon or definer). So, we defined it with «compiler'.
A small example, with cats and dogs of different types.

```
    2 constant dogs  10 variable cats       \\ Build them for this example
     : ToDogs ( n --)  >K dogs ;            \\ These are examples for Command line use
     : ToCats ( n --)  >K cats ;            \\ We would never use "ToDogs"  but rather "dogs !"
    3 ToDogs   7 ToCats                     \\ cats is a constant, we would use "ToCats"
    dogs .  cats @ .                        \\ ... it works!
```

Changing a constant may be useful. It should be practical but rare, speed is no concern.
We would want it to work on the Command-Line, not just compile-time... But will slow TO.
We'll need a OnCmdLine check: When is it running, were is it running ? That is imperative:

```
\\ Original 'TO' could be named  ' !: ' (it changes both Vars and Konst) to allow Values new 'TO'
    : TOK   I  1300  -      \ WordBody instead CmdLine?
      IF R>                 \ Yes, it's running from a word -> get caller
        dup 2+ >R @         \ advance, leaving xt
      ELSE   find           \ No, it's called from CmdLine -> get xt ...
                            \ May add a test to found (CmdLine has no need for speed)
      THEN 2+ ! ;           \ xt now pfield. Actualize its content
```

Konstants are very fast. If rarely written, using Konstants as a Value can be useful.
Both >K and TOK are slower than variable Fetch/Store. WHY use them? WHEN use them?
Can be as wheels when to learn driving a bicycle. For a trial (or as an hack).
... TO behaves as the later ANSI standard (reason for renaming it as TOK).

```
HEX Code TOK  \ Total =47 Bytes
          \ T=312 clks .\ Similar to '!'
            21c1 , 0514 , edaf , 2842 , 13  c,
            6069 , 0303 , 4ec5 , 4623 , 0303 ,
            60df , 7369 , 7223 , e9fd , cdc5 ,
            063f , 1a0e , 4ecd , 1808 , eb  c,
          \ CodeSize is #38 ... #9 (Head)
```

So far, so good... On the ACE this TO word is relatively very fast, close to '**!**' (store).
Redefining the hack as a primary word, is hard. Its success remind us a forgotten detail:
Postfix references do not use the stack: The postfix reference is compiled in the caller!

Problem is, if a problem at all, we may need to do the same with different data types.

Addenda: To build these and other kind of system words, a few tools may be needed:

```
  \ cFlags delivers more than just STATE (must be extracted)
    : ?EDIT   15422  c@  4  AND  ;   \\ cFlags bit3 is Compile/Edit mode (vs LineRun )

  \ Where are we running? WordBody (0) or CmdLine (1) ?!?
    : ?LINE   I'  1300 -  0=  ;      \\ Tried 1273, but 1300 is closer
```

A choice of Value

Here we have to consider the advantages of VALUE. With a speed similar to variables,
VALUEs are self-managed in the sense we can build a much needed 2Value (a double).
Are quite useful as counters, also as Floats storage (or doubles), easing work.

We built a different TO (for all the following) VALUE, 2VALUE and cVALUE.
Easier to use and versatile, are less prone to mistakes: The Value type does its job.
VALUEs can be very useful variables (our next task), tough CONSTANTs cost as much as a DROP.

VALUE entities are more as variables acting as Constants (2.2 slower than Constants)
We defined **TO** and also added **+TO** (allowing to increment a VALUE as +! does with Variables).
Also an utility **TO_** to reset a TO order, if needed to avoid surprises after a mistake:

Defining VALUE, we need a set of selectors (exceptions to natural behaviour):

1) **TO** Request to store value on stack into Value (Integer) or 2Value (Double|Float) types
2) **+TO** Request to increment the Value or 2Value object... Do NOT use it on a Float type !
3) **TO_** Simply resets a previous request, before Value invocation (An Oops! word)

With VALUEs, 'TO' is no longer a modifier of a following reference argument (not on stack).
'TO' word leaves a signal, ordering the first VALUE entity listening and (resetting it).

First, a Runtime Primitive (Checks ToFlag. Then either Fetching, Storing or Incrementing):
    hex code **(val)**   #43 Bytes with 3 Operators (available on the Primaries Library)
    The 'Definer' will use it to become a primary, much faster (and smaller):
    hex definer **VALUE** , DOES> (val) ; for regular 16bit Integers (also on the Library)

Later, we may define a 32bits 2Value. For now, we leave it as an exercise.

About speed
    Value is 2.2x <u>slower</u> than a Constant... It does not replace true constants.
    Value definer carries 3 methods: Read, Write, Increment... Thus, selection is added.
    So, the real question is HOW does it compares with variables. It depends on the Method:

Examples of usage

```
2 VALUE v.cats  \\ Builds a VALUE item containing '2' (for easy consultations)
v.cats          \\ here delivers '2', [ <val> ] only 1.06x faster than [ <var> @ ]
3 TO  v.cats    \\ Place 3 on v.cats, [ TO <val> ] only 1.21 slower than [ <var> ! ]
5 +TO v.cats    \\ Increment of 5, [ +TO <val> ] is now 1.53 faster than [ <var> +! ]
```

Were to use them?
As variables (mostly consulted, rarely written). Or to reduce code:
Then using these for increased readability. Surely as counters... Choose!
TO and +TO also work with Double Values. These simplifying stack management:
The best aspect maybe 2Value: 2VALUEs reduce double entries management (as variables).

After defining our 2VALUE with zeros (advised for simplicity), we will actualise them!:

```
1.23456  TO  v:width                \\ We use 'v:' for Doubles ('f:' for Floats)
12.3456  TO  v:height               \\ Here we 'init' two objects previously defined
```

Our stack management is now much simplified:

```
v:width  v:height  F*               \\ Simpler invocation before a Float multiplication,
TO v:result                         \\ result also kept on 32bits storage (keep stack clean)
```

Note: 2Value is still missing, for now (unless you build your own, as you may)

___
Again: More than tools, we need more RAM. Would need ROM tables (logaritmic and trignometic).
We may wait for a computer with a Math co-processor (a 286 costing a year salary, maybe) to play Tetris and Klondike (real world demands, by Department Directors, for the sake of status).

-

4.4 A Primaries Library

The missing, the crippled... and some useful extras

FORTH is very special, being a Language you can change. No other programming language allows that.
(...Usual languages are big, the only change allowed being added and renamed procedures.)

On ACE-FORTH, we can rebuild base instructions to load later, at will. Missing words can be added.
Crippled words can be replaced, shown in the following pages under the “no abuse” GPL-3 licensing.

TIPs for an immediate Speed-Up:
Using the patched ROM or not, or the recovered words, a major detail should be considered
for easy speed-ups: On the ACE, the Data-stack is slower than Return-stack.
This determines the following suggestions:

* On the ACE, **>R** and **R>** are fast:
  Each is 2x faster than a **SWAP** , and a fast way to keep local constants.
  Also: Moved values can be accessed with I, I' and J These Ops peeking the RStack
  when NOT using Do-Loop. IFF before a DO-Loop, only the last 'save' is seen (by J).

* **U/MOD** is 4x faster than **/** or **MOD**
  As long as values are not negative, we only have to drop one of the 2 values given:

```
    : U/    U/MOD DROP” ;       \ remove the mod result     ( NOT on the library )
    : UMOD  U/MOD SWAP DROP ;   \ remove the div result     ( NOT on the library )
```

POSTPONE word is here absent (not really needed on ACE-FORTH):
It is a slow Secondary, adapted to ACE-FORTH disassembly. Because slow, should not be used
on a small 8bits system. Latter I may change my solution into a Primary (most likely not).

The most important missing words are here present, plus a few replacements copyrighted as
"2021,Dutra de Lacerda" distributed under GPL-3 (as mentioned in the body of the listing).
All information is on the next pages. You decide what failing that agreement means
(The Full Library is divided into sections, but a single copyrighted work)
Please respect it, as any nice gift would be!

An example, again, of Hand Assembling a FORTH Primary, as Forth standard words are.
Restoring a Sec [: **2*** DUP + ;] back to the Primary glory, simply translating our Macros:

```
;;--------------    // 3.25x the Best Secondary (DUP +)
;; Code  2*         // FAST= 290 ... vs 940
;;--------------    // Note: 283 if using SHIFT (1 extra byte)
;  Dpop             ;89; RST 18h      ;1 ; df      \\ ebdf
;  DE+DE            ;04; ex  DE,HL    ;1 ; eb      \\
;                   ;11; add HL,HL    ;1 ; 29      \\ eb29
;                   ;04; ex  DE,HL    ;1 ; eb      \\
;  Dpush            ;89; RST 10h      ;1 ; d7      \\ d7
;  Next,            ;93; jp  (IY)     ;2 ; fd e9   \\ e9fd
;;                  ;-----------------------------;
;;                  =290              #7
```

Hexadecimal is much used to enter CODE words. That too makes the **HEX** word very important.
Here's a version that saves either bytes, either us corrupting the FORTH (vocabulary) name:

```
decimal 16 BASE c!  \ We assume you alredy have the CODE
CODE HEX ebdf , eb29 , d7 c, ENDCODE    \ BYTES used  Head=#9 + Code=#7  (saves 3 bytes)
```

Filling the Gaps

```
--- cut ---
: \ QUERY CR ; IMMEDIATE       \ comments (to discard, not compile)
\ ==========================================================
\ Descript: Small Lib, GPL3 licence, ©2017, Dutra de Lacerda
\ A small set for the restoration of ACE-FORTH Primary words
\ not fitting in 5K available. Some still absent, some added
\ ==========================================================
\ Usage: Copy file to SPOOL.TXT .. Alter SPOOL.TXT work file
\ Advice: After loading the copy, create your own V-USR voc.
\ Then, a USR: access IMM word as shown with V-UTIL (bottom)
\ ==========================================================
FORTH DEFINITIONS   \ on FORTH main branch
0 COMPILER [FORTH] FORTH RUNS> ;            \ Return to FORTH context on compiling
vocabulary :UTILS:                          \ Make VOCs visible,sight identifiable
0 COMPILER [UTILS] FORTH :UTILS: RUNS>  ;   \ Jump to :UTILS: context on compiling

\ Essential FORTH words to define missing words. Or to recover those turned Secondaries
: CODE create  here  dup 2- ! ;             \ Do not forget to choose: Hex or decimal ?!?
: END-CODE [ 16 base c! ]  e9fd , ;         \ Do not forget to choose: Hex or decimal ?!?
 CODE HEX  [ 16 base c! ]  103e ,  32 c, 3c3f , e9fd ,      \ HEX as a Primary is smaller (!)
 CREATE R@   find I @    here 2- !          \ A faster bodyless definition. And yet, a Primary)

\ *** FILLING THE GAPs -- First, 8 Words available again ***
FORTH DEFINITIONS   \ on FORTH main branch

 HEX CODE  +!       \ 5.5x Best Secondary   \ Much sed, this word should be on Rom
  d5df , e1df , 234e , 2b46 , 09eb ,
  73eb , 7223 , End-Code

 HEX CODE  2*       \ 3.2x the best Secondary (Use SLIDE to build D2* as secondary)
  cbdf , cb23 , d712 , End-Code             \ Much used, this word should be on Rom

 HEX CODE  2/      \ 46x the Secondary (Use SLIDE- to build D2/ as secondary)
  cbdf , cb2A , d71B , End-Code

 HEX CODE 2DUP      \ 12.3x (OVER OVER)
  3b2a , 543c , 015d , fffc , 0109 ,
  0004 , b0ed , 22eb , 3c3b , End-Code

 HEX CODE FILL      \ 19.7x the Secondary
  7bdf , 42df , df4b , 5feb , b079 ,
  0528 , 2373 , 180b ,  f7 c, End-Code

 HEX CODE CMOVE     \ *** OK ***
  42df , df4b , dfd5 , d1eb , b178 ,
  0228 , b0ed , End-Code

 HEX CODE CMOVE>    \ *** OK ***
  7adf , 20b3 , df04 , fddf , 1be9 ,
  4b42 , ebdf , e509 , ebdf , d109 ,
  ed03 ,  b8 c, End-Code

 HEX CODE MOVE      \ *** OK
  42df , 7a4b , 20b3 , df04 , fddf ,
  dfe9 , dfd5 , e5e1 , 52a7 , 38e1 ,
  0b09 , eb09 , 0309 , b8ed , e9fd ,
  edeb ,  b0 c, End-Code
```

Common Usage

```
\ Descript: Small Lib, GPL3 licence, ©2017, Dutra de Lacerda
\ ==========================================================
\ *** RESTORING other common use words back to Primaries ***

FORTH DEFINITIONS  \ on FORTH main branch

 HEX CODE COUNT      \\ 4.0x the best secondary
  d5df , d713 , 5ee1 , 0016 ,  d7 c, End-Code

 HEX CODE  -         \ 2.1x the original
  4ecd , df08 , a7eb , 42ed , d7eb , End-Code

 HEX CODE  ABS       \ 4.3x the original
  cbdf , 287a , af06 , 6f67 , 52ed ,
  d7eb , Fend-Code

 HEX CODE  <         \ 2.36x  the original
  cddf , 084e , 0021 , 1980 , 21eb ,
  8000 , af09 , 52ed , 1757 , d75f , End-Code

 HEX CODE  =         \ 3.0x  the original
  4ecd , df08 , ebaf , 42ed , 7c57 ,
  feb5 , 7a01 , 5f17 ,  d7 c,  End-Code

 HEX CODE  MAX       \ 10.4x the original       \ Rarely used, would not be on Rom when finished
  cddf , 084e , 21d5 , 8000 , eb19 ,          \ Estimated days to finish Rom: ~3 days (better if ~5)
  0021 , 0980 , edaf , d152 , 0238 ,          \   ( Estimations based on our own work )
  5059 ,  d7 c, End-Code

 HEX CODE  MIN       \ 8.9x the original        \ Rarely used, would not be on Rom when finished
  cddf , 084e , 21d5 , 8000 , eb19 ,          \ Estimated days to finish Rom: ~3 days (better if ~5)
  0021 , 0980 , edaf , d152 , 0230 ,          \   ( Estimations based on our own work )
  5059 ,  d7 c, End-Code


\ *** CodeSize SAVERS -- Much used to reduce code size -- Good engineering ***
    \ The following words compile Half size, no (Lit). Are also a bit faster
    \ Note: The ‘0’ word only needs an header (lost in the battle), body still there.
FORTH DEFINITIONS  \ on FORTH main branch, as these should be always visible
 HEX CREATE 0   068a   here 2- !            \ 1.3x faster ( Header might be on Rom when/if finished )
 HEX CODE   1   0111 , d700 , End-Code      \ 1.3x faster, not as critical as <zero> still much used
    \ The following are fully optional, thus kept commented:
\\ HEX CODE   2   0211 , d700 , End-Code    \ 1.3x faster, cell size. It may be very used, or not
\\ HEX CODE  -1   ff11 , d7ff , End-Code    \ 1.3x faster-1. negative number, not “1-” decrementer

\ DEBUG DEFINITIONS     \  Fully optional, thus kept commented:
\\  : ?  ( addr -- )  @ . ;         \\ Check a Variable (integer  content)
\\  : u? ( addr -- )  @ u. ;        \\ Check a Variable (unsigned content)
\\  : t? ( addr -- )  COUNT TYPE ;  \\ Check TEXTvar (a counted Array, as PAD)
\\  : ?s ( start limit -- start limit)  OVER OVER TYPE ;  \\ Check (not named) STR(params)

\\ Are we Editing?  Editor (4) or Otherwise(0) ?!?
\\  : ?EDIT   15422  c@  4  AND  ;   \\ cFlags bit3 is Compile/Edit mode (vs LineRun )
\\ Where are we running? CmdLine (1) or WordBody (0) ?!?
\\  : ?LINE   I'  1300 -  0=  ;      \\ Is this CmdLine?
```

Essencial words

```
\ Descript: Small Lib, GPL3 licence, ©2017, Dutra de Lacerda
\ ========================================================
\ Stack words are Prioritary *** We must restore crippled STACK words to their original speed
FORTH DEFINITIONS       \ on FORTH main branch

 HEX CODE  OVER         \ 2.5x the original
 \ Here 3 bytes bigger than the Secondary on the original ROM
 \ Not to load IFF already using the Patched +CODE ROM
  d5df , 42df , d74b , d7d1 , 5950 , d7 c, End-Code

 HEX CODE  ROT          \ 2.1x the original
  d5df , d5df , 42df , d14b , d1d7 ,
  50d7 , d759 , End-Code

 HEX CODE  ROT-         \ 2.1x the Best Secondary
  4ecd , df08 , dfd5 , 50d5 , d759 ,
  d7d1 , d7d1 , End-Code

\ ... Plus 2 stack words, now ANSI
 HEX CODE  NIP          \ 2.2x (( swap drop ))
  d5df , d1df ,  d7 c, e9fd ,

 HEX CODE  TUCK         \ 3.5x the Secondary
  4bdf , df42 , 59d5 , d750 , d7d1 ,
  5059 ,  d7 c,  End-Code

\ Usefull ( and needed on game development )
 HEX CODE  LSHIFT       \ (16bitsVal, disp -- Val')
  43df , cbdf , cb23 , 1012 , d7fa , End-Code

 HEX CODE  RSHIFT       \ (16bitsVal, disp -- Val')
  43df , cbdf , cb3a , 101b , d7fa , End-Code

 HEX CODE  @EXECUTE     \ 10.7x the secondary
  ebdf , 235e , 7a56 , 28b3 , c303 ,
  04bf , End-Code


\ Random not being standard, it is important enough to be on FORTH main voc.
\ Since the example on the ACE Manual is slow, not practical enough, we implemented Marsaglia's
\ XOR-Shift algorithm for quick results.(and chosen "Magic Numbers" for good ‘random’ qualities).
\ Obtained both generation speed of ~9/msec (8800/sec) and a small size of 10+39= 49 bytes(!)

 HEX CODE  RND      \ RANDOM values range is [1..64k]   (it never ouputs a zero!)
  372a , e53c , 235e , eb56 , b57c ,
  0220 , AA2e , 1f7c , 1f7d , 3e57 ,
  1f00 , 5fad , ac7a , 1f57 , 5fab ,
  57aa , 73e1 , 7223 ,  d7 c, End-Code

\   Its Seed is pointed by the HERE word.   \ Seed can be changed with    n HERE !
\   where (n) is the seed new value...      \ Seed can be consulted with    HERE @
\   If needed, or having enough RAM, we can then build the pseudo-variable SEED as:
  : SEED  here ;     \ Pseudo Variable, 16bits.  On invocation it delivers ( ---addr )

\ Similarly, on the Jupiter we usually use I to invoke R@  Having enough Ram, have build it as:
\ CREATE  R@   FIND I  HERE 2- !            \ Its code is I   The R@ header did not fit on Rom
                                            \ Restoring it we keep R@ as a primary (and shorter)
```

Strings & Speedups

```
\ Descript: Small Lib, GPL3 licence, ©2017, Dutra de Lacerda
\ ============================================================

\ *** OTHER non standard words, adapted fro the standard to ACE editing functionality:
FORTH DEFINITIONS                    \\ From FORTH-83 Standard, here hacked for the ACE
\ Inline Strings, similar to a  ."  are available with our hacked S" ( string as as comment)

 HEX CODE (s")                       \\ S" RunCode, to be used on the COMPILER definition.
  13df , eb13 , 235e , 2356 , ebd5 ,
  d1d7 ,  d7 c, e9fd ,               \\ End-Code already inserted
 0 COMPILER  S"                      \\ S" followed by comment immediately after!!!
    RUNS> (s") ;                     \\ run code extracts string data from the comment. Done!

 : Cnt2t  ( OArray DArray --)        \\ Simple Counted Char Array... to Counted Char ARRA
     over c@ 1+ cMOVE ;              \\ !!! Safe Destination size test is missing

\ Its pair ( also named >COUNTED )   \\ It can be hard to choose non-equivocal names
 : Str2t ( Ostr length Darray -- )   \\ ChArray(Params)... To Counted CharArray
   >R dup I c!      \ Ostr len          || Darray(with size)
   R> 1+ swap       \ Ostr Darray len len  || -
   cMOVE ;                           \\ Note that safe size test is not done here either

\ *** OTHER non standard words that SPEED-UP, EASE, and CLARIFY code ***
\ These to place on :UTIL: . These words should be on any standard FORTH !
FORTH  DEFINITIONS                   \\ Not Standard, though they should
\ USAGE: <var> <increment> ..cleaner and faster than.. N <var +!

 HEX CODE  1+!       \ ~2.4x(  1 +! )      \ Similar to <var>++ in C
    d5df , 5eeb , 5623 , 13d7 ,            \ INCrement a <var>
    73e1 , 7223 , End-Code

 HEX CODE  1-!       \ ~2.4x( -1 +! )      \ Similar to <var>-- in C
    d5df , 5eeb , 5623 , 1bd7 ,            \ DECrement a <var>
    73e1 , 7223 , End-Code

 HEX CODE  CELL+!    \ ~2.4x(  2 +! )      \ Similar to <index>++ in C
    d5df , 5eeb , 5623 , 13d7 ,            \ ... INC by CellSize (was 2+!)
    13  c, 73e1 , 7223 , End-Code

 HEX CODE  CELL-!    \ ~2.4x( -2 +! )      \ Similar to <index>-- in C
    d5df , 5eeb , 5623 , 1bd7 ,            \ ... DEC by CellSize (was 2-!)
    1b  c, 73e1 , 7223 , End-Code

\ 16 bit-Rotation, similar to Z80 8 bits RL "reg"  (Use SLIDE to build signed 2*)
 HEX CODE  SLIDE ( carry', value'  --- carry", value" )
    4ecd , df08 , 1f7b , 11cb , 10cb ,
    001e , 13cb , 50d7 , d759 , e9fd ,

 HEX : HASH ( Hash, Data --- Hash")        \\ Our 16 bits fast Hash
    >R  0 swap Slide  +     \ Rotate Hash = Add Slide Value to the new carry
    R>  xor  352b +   ;     \ xor Hash' with Data, then add Disperser

\ ==========================================================
\ End of restoration and added words. Return to FORTH root if elsewhere.
FORTH  DEFINITIONS  Decimal                \\ Our exit from Lib code
--- end-cut ---
```

DOUBLEs extensions

```
\ Descript: Small Lib, GPL3 licence, ©2017, Dutra de Lacerda
\ ========================================================
\ *** EXTENSIONS to be loaded occasionally,  when needed ***
\ On the ACE Doubles would be mainly used for Floating Point
\ Not limited there, his Primaries speed up double arguments

 HEX CODE  2DROP    \ (fp1 — )  aka (1,2, 3,4  —  1,2)             (( DROP DROP ))
  dfdf , e9fd ,

 HEX CODE  2DUP     \ (fp1,— fp1,fp1)  aka (1,2  — 1,2, 1,2)       (( OVER OVER ))
  01af , 0004 , 5bcd , 0109 ,
  0004 , 5bcd , 09  c, e9fd ,

 HEX CODE  2OVER    \ (fp1,fp2 — fp1,fp2,fp3)                      (( 4 PICK 4 PICK ))
  01af , 0008 , 5bcd , 0109 ,
  0008 , 5bcd , 09  c, e9fd ,

 HEX CODE  2SWAP    \ (fp1,fp2 — fp2,fp1)  aka (1,2, 3,4  — 3,4, 1,2)    (( 4 ROLL 4 ROLL ))
  4ecd , df08 , dfd9 , dfd5 ,
  d9d5 , 50d7 , d759 , d1d9 ,
  d1d7 , d9d7 , e9fd ,

 HEX CODE  2NIP     \ (fp1,fp2  —  fp2)  aka (1,2, 3,4  —  3,4)    (( 2SWAP 2DROP ))
  4ecd , df08 , dfd9 , d9df ,
  50d7 , d759 , e9fd ,
```

\ *Fair Note: With the exception of the following quick implementation, all the above words are*
\ *MUCH FASTER than its size increase... in face of their secondary equivalents (due Soft-Stack).*
\ Following 2ROT is also faster, but bigger. It can have a better speed/size ratio. Rarely used.

```
 HEX CODE  2ROT     \ (fp1, fp2, fp3 — fp2,fp3,fp1)                (( 6 ROLL 6 ROLL ))
  d5df , d5df , d5df , d5df , cdd9 ,
  084e , d9df , d7d1 , d7d1 , d7d1 ,
  d7d1 , d7d9 , 5059 , d9d7 , e9fd ,
```

\\ Warning: Many 8bit Forth's implemented storage in reverse, as CPU's extension OPs loaded 16bits.
\\ This was an odd choice suggested by Little Endian 8bit Architectures, extended out of its scope.
\\ *We chose to implement storage (of Doubles) as 32bits, disregarding how 16bits is stored.*
\\ *Work the same. Consistency is kept, is more compatible with native 16 and 32 bits*
\\ *Also allows a direct BCD Floats DUMP view,* as storage returns to Big-Endian.

```
 HEX CODE  2!       \ ( xL, xH, Faddr — )                          (( DUP >R ! R> 2+ ! ))
  d5df , cddf , 084e , 73e1 ,
    7223 , 7123 , 7023 , e9fd ,

 HEX CODE  2@       \ ( Faddr — xL, xH )                           (( DUP 2+ @  SWAP @ ))
  ebdf , 234e , 2346 , 235e ,
  d756 , 5950 , d7  c, e9fd ,
```

Non standard

```
\ Descript: Small Lib, GPL3 licence, ©2017, Dutra de Lacerda
\ =========================================================
\ *** NOT STANDARD but very useful, also w/ SYSTEM WORDS ***
\ To be implemented only when needed, placed on :UTIL: vocab

\\ ** REMINDER **  Prepare changes of VOC context *also* on compiling mode:
\\      FORTH  DEFINITIONS                        \ FORTH Voc already exists
\\      0 COMPILER [FORTH] FORTH RUNS> ;          \ Now a return when compiling
\\ For every *new* Voc , follow the example of the following 2 lines:
\\      vocabulary :UTILS:                        \ in between colons makes Vocs visible
\\      0 COMPILER [UTILS] FORTH :UTILS: RUNS> ;    \ Immediate context change, runs nothing
\\ Definitions now can use ... [UTIL] <libword> ... on compiling!
\\ To be followed by a Return to their Voc. Do NOT forget were your words are being defined!
\\
\\ ** FAQ **  WHY to do the preparation above? - Because Context is actual (Voc-Search is not)
\\ An example of a foreign VOC invocation could also be:   [UTIL] <size> wArray <name> [FORTH]
\\ This changed context to :UTIL:  called a definer there, created an array here.
\\ We then return to the FORTH context (it could be another). This strategy allows easy VOCs,
\\ easily invoked in compiler mode... In spite the absence of an actualized search-path-list.

\ Check Space Key Pressed, indicating a SoftBreak request (no ABORT).
 :UTIL:  DEFINITIONS
 HEX CODE  spKey?     7f3e , fedb , 01ee , c31f , 0c21 ,     \\ Is Space Key pressed?  Avoid Break
\ Usage: [ UTILS] SpKey? [ FORTH ]  IF ... THEN  ...         \\ Can be used to quit a long process

\ SP! is a System-Utility word, defined as ACE-FORTH Secondary. Rarely used, it goes into :UTIL:
 :UTIL:  DEFINITIONS
 : SP!   ( cells -- ) dup + HERE + 12 + 15419 ! ;    \ inside a word... [UTILS] n SP! [FORTH]
\ Example of usage, here empting the stack (zero size):
\   ... [UTILS] 0 SP! [FORTH] ...
\ How does this scheme works? Go :UTILS:, compile a word there, then return to FORTH context

\\ BYTE cARRAY : Definer not realy needed. We define Byte Arrays as   CREATE <name> <size> ALLOT
\\ Simple arrays are simply Vars accessed  with   <name> <index> +   Simple and fast.
\\ Formal definition is   DEFINER cARRAY allot DOES>  + ;  used with ... <index> <name>
\\ While cARRAYS need no defining word, TEXT arrays demand it. ((To ‘size’ we added ‘limit’))

\ TEXT Counted cArrays, used for storing Text Strings, have an header, can defined as
:UTIL:  DEFINITIONS                     \\ Place following on :UTIL: vocabulary
    DEFINER tARRAY ( byte -- )          \\ warning: max is 253... we could test if <=253
      dup ,  allot                      \\ 2 Header bytes: limit, size. Followed by <reserved space>
    DOES>  1+  ;                        \\ ( -- addr )  Ignore our limit byte... Proceed as standard

\ WORD ARRAYS: ACE disassembly prohibits ;CODE ... So, we'll build runtime code words
:UTIL:  DEFINITIONS                     \\ Place following on :UTIL: vocabulary
    HEX CODE (2POS)                     \\ Index,Array->Addr (( SWAP DUP +  + ))
      4ecd , df08 , 23cb , 12cb , 09eb ,
      d7eb , End-Code
    DEFINER  wARRAY  2* allot  DOES>  (2POS)  ;         \\ ( index, wARR --- IntAddr )

\ ARRAYS of DOUBLES, or FLOATS.     \\ These deliver the item address (ready for 2!  and 2@)
:UTIL:  DEFINITIONS     \ As above, to load if needed
    HEX CODE  (4POS)    \ 3.14x Best use, on DOES>
      4ecd , df08 , 23cb , 12cb , 23cb ,
      12cb , 09eb , d7eb , End-Code
    DEFINER  dARRAY  2* 2* allot  DOES>  (4POS)  ;      \\ ( index, dARR --- FAddr )
```

ANSI extensions

```
\ Descript: Small Lib, GPL3 licence, ©2017, Dutra de Lacerda
\ ========================================================
\ *** EXTENSIONS to be loaded occasionally,  when needed ***


\ Time delay ANSI tool ( u -- ) ...  1 ms up to 64 seconds
 HEX CODE  MS                                    \ accurate MiliSecond delay
  06df , 10f8 , 1bfe , b27b , f720 , e9fd ,

\ In Range is an important part of decision making
 HEX CODE   WITHIN    \ ~7x best secondary        \ range identifier test
  d5df , 42df , e14b , afeb , 52ed ,
  dfd5 , 6960 , afeb , 42ed , afd1 ,
  52ed , 21c3 ,  0c c,

\ Translate F83/ANSI by its F79-ACE equivalent
 : CHAR  ASCII ;

\ Translate F83/ANSI by its F79-ACE equivalent
 : [CHAR]  [ find ASCII , ] ; IMMEDIATE

\ Build F83/ANSI (tick) as F79 FIND (F83 FIND is not ACE FIND)
 CREATE  '   find FIND   @   here 2- !          \ for F83 code conpatibility


\ ANSI formalises TO as a flag. We added a needed equivalent of +! (and a flag reset).
 HEX CODE  TO    013e ,  32 c, 2de1 , e9fd ,    \ This works with 2Value, can store Doubles     #=Header+7
 HEX CODE +TO    023e ,  32 c, 2de1 , e9fd ,    \ If storing a Float... DO NOT increment it!
 HEX CODE  TO_   003e ,  32 c, 2de1 , e9fd ,    \ Oops!... Turn the TO flag off ("or else!")

\ VALUE data type are variables (actualised differently) while behaving as constants.
 HEX CODE  (val)  \ #43 Bytes                   \ Turns Value into a Primary word               #=Header+43
  3adf , 2de1 , af08 , 32  c, 2de1 ,            \   A small Note on its size:
  b708 , 0720 , eb  c, 235e , d756 ,            \   On 43 bytes of code are
  e9fd , 4ecd , eb08 ,  3d c, 0520 ,            \   3 different Forth actions
  2371 , 70  c, e9fd , 235e , eb56 ,            \   and its selection on runtime
  eb09 , 2b72 ,  73 c, e9fd ,                   \   Check its characteristics and usage
DEFINER VALUE , DOES> (val) ;                   \ 16 bits data type                             #=Header+8

 HEX CODE (2val)    \\ MUST BE COMPATIBLE with 2! and 2@    \ It is left as an Exercise, for now
  e9fd ,            \ (this is just a NOP, for a while, to allow the following definition)      #=Header+xx
 DEFINER  2VALUE    \\ 32 bits data type        \ ie: Choices must be kept CONSISTENT!          #=Header+12
  , ,               \ simpler built (though irrelevant): SWAP no longer needed
  DOES> (d2al)  ;   \ On Stack HI is on Top for quick check and correct Print
```

Chapter 5 - The ACE ROM Project

5.1 - So, What's on ROM?

```
BIOS    (~3K)

*   INIT:
    Boot        =  269          \ System init + Z80 vectors
    SVini       =   45          \ System Vars init
    ChDef       =  641          \ Video Chars init
               =======
                  955 Bytes

*   SYS:
    VSynch      =   68          \ KBD timer and Stack Sanity Check (Safe mode, aka SLOW)
    Console     =  710          \ CP-M usuals + (small) Number Conversion
    TapeCode    =  724          \ Replacement of CP-M Disk Access
               =======
                 1502 Bytes    \ Most usually

*   Libraries:
    Float Lib   =  647          \ External to traditional Forth
               =======
                  647 Bytes    \ A very condensed set


FORTH   (~5K)

*   CORE:
    Kernel      =  965          \ Dispatcher + Routines
    4thWords    = 1689          \ Stack + Logic + Arithmetic
    Compilers   = 1351          \ Control Structures + Interfacing
               =======
                 4005 Bytes    \ include near 1K of Word Headers

*   EXTRAS:
    Decompile +                 \ Fetch and identify routine
    + Edit + List +             \ List is Edit on Read-Only mode
    + ReDefine  = 1070          \ Reallocate, for replacement insertion
               =======
                 1070 Bytes    \ These are added to traditional Forth
```

Added Note: A few code locations (for a quick find)

```
; 013a  - Init, Tables and Services            \ ~1100 Bytes
; 049d  - Forth (with decompilation extensions) \ ~4005 Bytes
; 13fd  - Specials (Ace tools + decompiling)    \ ~1070 Bytes
; 1820  - Tape SubSystem                        \ = 724 Bytes
; 1af4  - Floats Library                        \ = 647 Bytes
; 1d7b  - Char definitions                      \ = 641 Bytes
; 1ffc  - Link (Lost © word header)             \ =   4 Bytes
```

5.2 Sections Overview

True sizes of ACE-Forth ROM sections, are no less than amazing:

```
              ~0.7K ... a very small 32bits FLOATS LIBRARY (!!!)
              ~2.5K ... is  SYSTEM = (BIOS +Tape +INIT +CharTable)
          Making a total of ~3.1K, out of those 8K, that should be on an external 4K ROM
          This would allow the use of a 1K trigonometry table for welcome quick results.

          (Leaving ~5.6K to Forth and Extras)
              ~4.0K ... is  FORTH = (KERNEL +DICT +COMPILING)
              ~1.0K ... to EXTRAS = (LIST/DECOMPILE +EDIT)
```

Reminder: Sys UTILs being TAPE routines ... Extra UTILs being DECOMPILE+EDIT
These are FORTH+Extras = ~5.0K (of which >1K are headers), with Decompiling satisfied.
(Near half of 1.3K 'Compilers' code are a small extra also used for editing).

Reminder: ROM should be on the end of Address Space, so to allow access and change of RSTs.
But also, to ease the move from a CP/M machine, instead a painful one at a time add.
We have experienced the problem of too different systems, while patching the ROM.

We have experienced a similar demand. Refrained to slowly do only one change at a time.
Much time is wasted, aggravated by a wrong architecture. There, a little time spent
rebuilding the ACE with CPUstk would allow to save much more time (lost on debuging).

.**In retrospective:**.

* A feat needed: It would take a few years for diskette drives to become affordable.
* But it ALSO allows changes without a source reload. That was unique.
* Nor repeated, though that kind of convenience was appealing.

A few more considerations:.

- Full Forth + Specials would be 6+1=7K   (BIOS needed its own slot, away)
- Specials, because system related could be on the absent System ROM, of 4K.
- Then the ACE could be full, optimised, Multi-Tasking. With a safe 2K spare.

The above, is just a thought! Instead, it also dropped from 20xBasic to 15.7x
At the the incorrectly measured as 12.4x (or 11.6x, presented as a sure 10x).
And yes, redesigned it would be 35x faster (Sieve of 45x). That, and 12K.

Remind bench distortion was caused by **OVER** and by **<** , more than using FILL
as a secondary (a FORTH word, a primary), as foreign procedure.added.
Anyway, removing FILL gives a closer result (also benefits BASIC).
Weights measures confirmed all this. ROM patches prove it (!).

In Short:

- **System =** ~<u>3116</u> Bytes 3K
- **Forth   =** ~4005 Bytes 4K
- **Specials=** ~1070 Bytes 1K

"This ACE, can develop!" ... Though hardly with only 900 bytes RAM.
Even if a 3.5K FORTH (4K - half of Compiling). Needed 4K dynamic RAM not tested.
Nor translated into the handmade blueprint... The blueprint, a damned limitation.
The hardware never paired the firmware. Does not matter why, matters to know why.

5.3 System Structures

Please note:

This is a system reference, not needed to neither regular usage.
Some later changes were not actualised on the Listing (were just comments).
Thus this more actual description of some structures, after the final ROM:

```
ROM-WORD:  Position    Meaning
           -n-4    ..  Name in ASCII, Last Char has Bit 7 = 1      (ROM=n-4 RAM=n-6)
            -4     DW  Not needed nor present on ROM to save space.(Must check ROM or RAM)
            -2     DW  Link to Previous Words. Used by FIND
             0     DB  n=Name Lenght (bit6=IMM). Entry point for above Word Linkage
               +1  DW      Code Field Address (points Action or TypeAction)
                   ..      Parameters (Words Type Data)

RAM-WORD:  Position    Meaning
           -n-6    ..  Name in ASCII, Last Char has Bit 7 = 1      (ROM=n-4 RAM=n-6)
            -4     DW  Number of Bytes to End-of-Word, on RAM only.(Must check ROM or RAM)
            -2     DW  Link to Previous Word. Enforces Last-In-First-Found
             0     DB  n=Name Lenght (bit6=IMM). Entry point for above Word Linkage
               +1  DW      Code Field Address (points Action or TypeAction)
                   ..      Parameters (Words Type Data)


VOC-WORD:  *** Here correct, on the Listing description failed to be actualised  ***
           -n-6    ..  Name in ASCII, Last Char has Bit 7 = 1
            -4     DW  Number of Bytes to End-of-Word
            -2     DW  Link to Preceding Word in Parent's Vocabulary
             0     DB  n=Name Lenght (bit6=IMM). Entry point for above Word Linkage
               +1  DW      CFA = Set CONTEXT with the following parameter
                   DW      Last Word in this Vocabulary
                   DB      Flag, always 0... Entry point (not used) for above PrevVoc Linkage
                   DW      Ptr to Prev_Voc (List no longer used) Similar to ^PrevWord.
```

DATA STORAGE

```
    INTEGER:    2 BYTES     Placed in the Little-Endian way (Low, High)

    FLOAT:      3 BYTES     MANTISSE in BCD format
                4th BYTE    EXPONENT (-40H..40H), bit 7=Mantissa_SIGNAL

    TEXT:       ( As in Turbo-Pascal, are counted arrays. More efficient than end flaged )
                1 BYTE      SIZE (can be zero, data chars following)
                N BYTES     The String itself, up to 255 Bytes
```

RAM SPACE after Booting

3000h (4K) RAM space partially used, actually starting at **3C00h**

```
    3000h \
    3400h  > Absent 4K dinamic RAM
    3800h /
```

3C00h (1K static RAM)

```
        This area is initialized at Boot. First 64 bytes are SysVars
        3C00h : SYSVARS
        3C40h : FORTH vocabulary link
        3C??h : User dictionary, plus 12 bytes GAP before moving STACK
```

End-Of-Available RAM

SYS_Variables (console)

Please note:
This is a system reference, not needed to neither regular usage.

(based on the English disassembly listing)

```
; ---------------------
; THE 'SYSTEM VARIABLES' - Mainly Console (includes HOLD and EDIT vars)
; ---------------------
; "Here is a list of system variables. We have given them all names,
; just for ease of reference. The ACE will not recognize these names,
; except for a few. As 'BASE', CONTEXT AND CURRENT (FORTH words).
;
; FP_WS       $3C00 (15360)   19 bytes scratch workspace.
;                             Used by floating point arithmetic.
;
; LISTWS      $3C13 (15379)   5 bytes scratch workspace.
;                             Used by 'LIST' and 'EDIT'.
;
; RAMTOP      $3C18 (15384)   2 bytes - the 1st address, above RAM used.
;
; HLD         $3C1A (15386)   2 bytes. The address of the latest character held
;                             in the pad by formatted output. ('#', 'HOLD' ...).
;
; SCRPOS      $3C1C (15388)   2 bytes. The address of the place in video RAM
;                             where the next character is to be printed
;                             (i.e. the 'print position').
;
; INSCRN      $3C1E (15390)   2 bytes. The address of the start of the
;                             current 'logical line' in the input buffer.
;
; CURSOR      $3C20 (15392)   2 bytes. The address of the cursor in the
;                             input buffer.
;
; ENDBUF      $3C22 (15394)   2 bytes. The address of the end of the current
;                             logical line in the input buffer.
;
; L_HALF      $3C24 (15396)   2 bytes. Address start of the the input buffer.
;                             The input buffer itself is stored in the video RAM.
;
; KEYCOD      $3C26 (15398)   1 byte. The ASCII code of the last key pressed.
;
; KEYCNT      $3C27 (15399)   1 byte. Local to the routine that reads the kbd.
;
; STATIN      $3C28 (15400)   1 byte. Local to the routine that reads the kbd
;
; EXWRCH      $3C29 (15401)   2 bytes. Alternative ECHO routine. Usually zero.
;                             If non zero, replaces standard echo-to-screen.
;                             Available to echo a char to a different dest.
;
; FRAMES      $3C2B (15403)   4 bytes. These four bytes form a double length
;                             integer that counts the time since the Ace was
;                             switched on in 50ths of a second.
;
; XCOORD      $3C2F (15407)   1 byte. The x-coordinate last used by 'PLOT'.
;
; YCOORD      $3C30 (15408)   1 byte. The y-coordinate last used by 'PLOT'.
;
```

(Continues)

SYS_Variables (FORTH)

Please note:

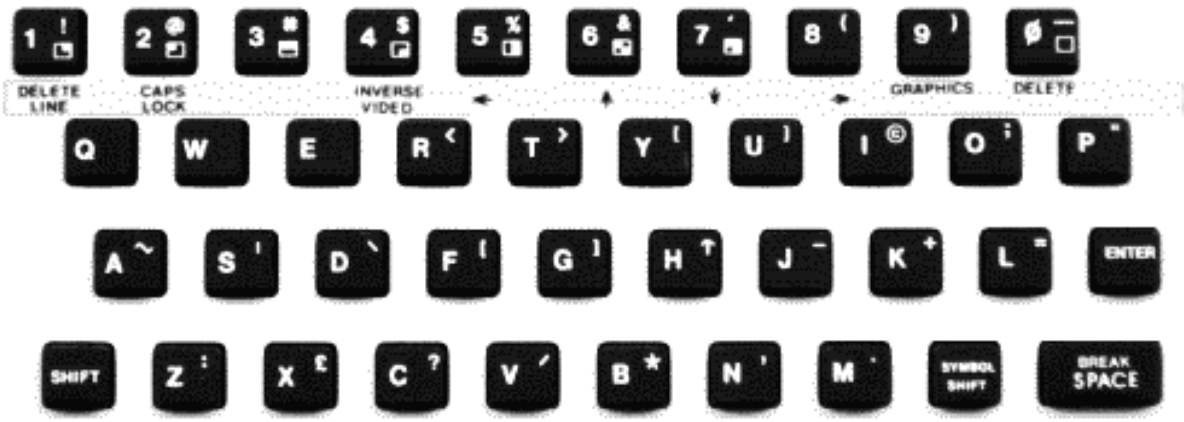
This is a system reference, not needed to neither regular usage.

(Continued)   (based on the English disassembly listing)

```
; ---------------------
; THE 'SYSTEM VARIABLES' - FORTH inner variables, mainly DICTIONARY
; ---------------------
;
; CURRENT       $3C31 (15409)   2 bytes. The parameter field address for the       ( -- addr)
;                               vocabulary word of the current vocabulary.
;
; CONTEXT       $3C33 (15411)   2 bytes. The parameter field address for the       ( -- addr)
;                               vocabulary word of the context vocabulary.
;
; VOCLNK        $3C35 (15413)   2 bytes. The address of the fourth byte in the
;                               parameter field - the vocabulary linkage - of
;                               the vocabulary word of the most recently
;                               defined vocabulary. A base for MultiVoc search.
;
; STKBOT        $3C37 (15415)   2 bytes. The address of the next byte into
;                               which anything will be enclosed in the
;                               dictionary, i.e. one byte past the present end
;                               of the dictionary.
;                               'HERE' is equivalent to 15415 @.
;
; DICT          $3C39 (15417)   2 bytes. The address of the length field in the
;                               newest word in the dictionary. If that length
;                               field is correctly filled in then DICT may be 0.
;                               Without this flag nature it would serve as LAST.
;
; SPARE         $3C3B (15419)   2 bytes. The address of the first byte past the
;                               top of the stack.
;
; ERR_NO        $3C3D (15421)   1 byte. This is usually 255, meaning "no error".
;                               If 'ABORT' is used, and ERR_NO is between 0 and
;                               127, then "ERROR" will be printed out, followed
;                               by the error number ERR_NO.
;
; FLAGS         $3C3E (15422)   1 byte. Shows the state of various parts of the
;                               system, each bit showing whether something
;                               particular is happening or not. Some of these
;                               may be useful.
;
;                               Bit 2, when 1, shows that there is an incomplete
;                               definition at the end of the dictionary.
;
;                               Bit 3, when 1, shows that output is to fed into
;                               the input buffer.
;
;                               Bit 4, when 1, shows that the Ace is in
;                               invisible mode.
;
;                               Bit 6, when 1, shows that the Ace is in compile
;                               mode.
;
; BASE          $3C3F (15423)   1 byte. The system number base. Starts with $0A.       ( -- var)
;
; ---
```

Keyboard Access

Please note:
This is a system reference, not needed to neither regular usage.




```
;  ^                     LOGICAL VIEW OF KEYBOARD                        v
            ; PORT                            <-bits->                        PORT
            ;  ^     0     1     2     3     4   |   4     3     2     1     0     v
            ;  |  _____________________________________________________________  |
            ;  ^                                 |                               v
            ; F7FE  [ 1 ] [ 2 ] [ 3 ] [ 4 ] [ 5 ]  |  [ 6 ] [ 7 ] [ 8 ] [ 9 ] [ 0 ]  EFFE
            ;  ^                                 |                               v
            ; FBFE  [ Q ] [ W ] [ E ] [ R ] [ T ]  |  [ Y ] [ U ] [ I ] [ O ] [ P ]  DFFE
            ;  ^                                 |                               v
            ; FDFE  [ A ] [ S ] [ D ] [ F ] [ G ]  |  [ H ] [ J ] [ K ] [ L ] [ ENT ] BFFE
            ;  ^                                 |                               v
            ; FEFE  [SHI] [SYM] [ Z ] [ X ] [ C ]  |  [ V ] [ B ] [ N ] [ M ] [ SPC ] 7FFE
            ;  ^           v                                            ^        v
            ;   \          +------------>----(key moved)------>-------------+       /
            ;    \____________________________________________________________/
```

Scanning the Keyboard
These 8 Ports are read, one at a time. A pressed key deliver a 5 bits Value.

Notice the 8 Ports are selected by the High Address (8 bits, one bit for each row)
while the Low Address remain constant, $FE (bit0 identifying a Keyboard read).

A more efficient choice would be bit7, &7F, to allow more flexible I/O access
to Addons... also allowing shorter code when needed.

Available Mosaic Chars
These allow Low Video Resolution Graphics (64x48) to coexist with text.
Each Mosaic graphic is composed by 4 squares were each (#) may be set.
These big dots (on mosaics) are identified by the following bits:

```
 |1 0| corresponding to the values |2 1| summed.
 |3 2|                             |8 4|

Thus the Graphic #0 is |   |  the #7 is |X X|  then  #8 is |   | and #15 is |X X|
                       |   |            |  X|              |X  |            |X X|
```

Notice Graphics #8 up to #15 are the inverse video of #7 down to #0 (after bit3).

5.4 ACE ROM tweaking

• ACE ROM section has been moved to the restored ROM listing.
See it on Appendix... Moved there, as [1981 Original Listing].
Version 1.x ROM Documentation stratified disassembly moved
to my private Notes (not here) as they could be 'excessive'.

There's no reason to keep that work (v1.x, nor the missed v2.x). But also:
The disassembly used by then, of authors unknown was, still is, very useful.

• New entries to this One-Man Project

- Possible ROM changes, after a decades old "impossible". (Done and Shared)
  We give not 'the' solutions, but some hints. A solution 'given' to be seldom educative,
  rarely useful. People deserve better. Deserve to exercise, discover, and grow.

- Original VOC search system (not the code but the methodology).
  That search system is suggested by residual structures (of the removed).
  Such search is crucial to any programming language. There are 2 possible ways.
  There's also a compromise on Search, between no VOCs and the original Search LIST.

  While standard VOCs allow a powerful programming environment, ACE later replacement
  would also be as powerful and easier. This replacement used by the ACE was not finalised.
  Now re-designed, we may implement it. Maybe we will. We now have space, and we have time.

- Mentioned (on Programming Tips) some actions of DEFINER and COMPILER (some are shared).
  Beyond standard CREATE DOES>, these are foreign ideas, the ACE Author managed to implement.

  We limited to a description, not to unveil (nor to trash) what is available in plain sight.
  It's the ONLY implementation known of a unified, well factored process. Hard to follow,
  more difficult to implement (specially with a soft stack promoted by Z80 practices).
  ((We were lost in the abstraction. Imagine the pain to implement (Moore's idea?),
  ((to factor a Quadri-level unifying way to handle both classes and compilers.
  ((It achieved its goals! Complexity needed, very 'practical'. Successful.

- Moved to an "eXtra" chapter some of the (inner) technical details of Forth:
  IE: Dispatchers on different implementations and a brief reference to other 'modes'.
  The essence of compilation (not examined) is there, Forth easing optimisations.
  (There are very good articles on Forth Dimensions regarding native code.)

- Also added is our recent adventure on Patching the ROM:
  First claiming ROM space, then restoring some ACE-Forth Words (one patch is available).
  Later compressing the BIOS Chars Definition Table... supposedly not possible.
  (We give an idea of that endeavour on the following pages, as its results.)

  Then corrected the Benchmarks distortion with our shared patch, restoring the OVER word
  and an hidden word. (Another tweak is waiting to be released, without the developing
  CODE word. It's indistinguishable from the ORIGinaly published ROM, one of the reasons
  for a delay on its sharing. For the common USER, it is much faster -- see Benchmarks.)

All this showing the 'big' effects of 'small' details, as a BIOS out of place carrying
a huge price to pay in several ways. Resulted from not having the full ROM at the end of Memory
(as needed), this repeating one of the ZX-80 errors kept on later hardware. In particular,
it hardened the transition from a CP/M developing system to an archaic (by 1981).
Sadly, a limiting architecture (disrespecting the Z80 vectors scheme).
We all should be pleased by having those points no longer refutable (cleared, proven).

___
 # The world is a most delightful puzzle maybe to keep! When each where he does belong.
 # When all is destroyed, and little remains, The Universe knows, nothing is ever lost.

Before patching

Naturally, we have examined new ways to build the ACE.
We focused on direct dispatching, even on subroutine threading.
Our worry, however, was the 900 bytes we have available, instead 4K.

Due lack of RAM we have chosen the Direct method. 'Native' spends too much memory. 'SBR' is an option.
A major goal was to keep disassembly (no text sources). This way we lose speed, but we save memory.

We examined Camel first. Our design become much different due choices and diverse solutions.
It could not even call it ACE3 no longer. Later, got a unique dispatcher, even considering
the wide spectrum of Forth known solutions. Keeping decompiling and not affecting speed.

From 2 options, the ACE-Forth solution was the only not reducing speed. But with a disadvantage:
It forced the existence of a disassembly list of exceptions. Before its ‘reduction’ to fit 5K,
this list was possibly (surely) intended to copied to the User area (as a SysVars extension).

Our choice for disassembly was the other options. A bit slower, whatever the dispatcher.
Not too slow, it worked nicely at the expense of reformulating Dictionary management.
With it, Primaries are not affected, only Secondaries. *(An interesting exercise)*.

At the end it resulted a bit faster than Camel, while decompilable. Headers did increased in size.
(Actually, headers doubled in size, from 7 bytes plus name, to ~15 bytes+name. Code is elsewhere.)
It was completely unique (we recently found a partially similar). Ours is a complete solution.

Naturally all this was a delight. But 40 years after the Z80 and the 6502, is a useless curiosity.
It was nice to find problem, after problem, balancing them. That's the only way to really master
a subject. Look: Pre-built solutions are just ‘acceptance’. Never a mastering: a damned 'mask'.
!DO NOT copy nor stay on known levels! Thus, we will not give details (just hidden hints).

Ace-Forth 'modes' evaluations (on 3.25MHz vs BASIC/3.5MHz)

native	**direct**	**Indirect (traditional)**	batch

Real Ratios vs ASM (common workloads, of multiple-words)

	PL/I	Ace++	Camel	(ACE2)	PRIV	TOOL	BASIC	
weights	*(na)*	84.	87.	127	219	259	4235	**weights**
/ASM	*(na)*	**9.1**	9.3	**13.0**	23.6	**27.9**	**529**	/ASM

Sieve Ratios vs ASM (single procedure, of limited prims)

	PL/I	Ace++	Camel	(ACE2)	Priv	Tool	BASIC	
minutes	0.23	**0.87**	0.97	**1.74**	3.7	**5.0**	**79.6**	**minutes**
/ASM	2.1	6.2	6.9	12.5	26.3	35.6	615.0	/ASM

First thing disturbing, therefore a reason to check and recheck previous evaluations,
was the sudden disparity between SIEVE and the derivative WEIGHTs dependent values.
Suggesting 2 reasons, as understood. Reasons to leave differences as are, split:

1) SIEVE being a short set of words benefited by the new architecture.
2) WEIGHTS being wider, also reflects using multiple words (Enter-Exit).

These 2 reasons compound, accumulate. And hard to force a middle ground.
- Sieve reflects single procedure, more than its set limitation
- Weights reflects multiple procedures usage

The more each method makes more efficient Primary Words, the more nesting becomes
relatively more expensive. This is shown by Weights/Sieve on (Ace2 and Camel).
Also reason for Weights small difference (of Camel vs ACE++), bigger on Sieve.

Will ACE++ be implemented? There's no reason to. Then what? *(Finding Tweaks was that ‘what’)*
Then: Why not an independent version, for the PC? Would there be a benefit today? To whom?!?

Patching the ROM?

On getting space:
By 2008(?) someone on a semi-private talk mentioned a reduction of the char definitions table (to 6 lines/char) so get a bit more space. It seemed achievable but it always resulted on expending ~4 more bytes (thus the request). Conclusion was, the ROM saved those 4 bytes.

Here, we first focused on getting code smaller. Then examined compression, facing a similar result. Traditional compression was completely discarded, later noticing it was possible (paradoxically?) by not compressing at all. Painting chars were and when possible might required less code. (Note: There are no paradoxes, all being mirages born from interpretation. Same here.)

Not flashy code but humble and small (that's how we got space for the green, compressed tables ROM). Otherwise the code would be bigger than any space recovered. Smaller tables are needed to recover words dropped, to optimise others. Or to alter the internal behaviour of FIND (extra ~75 Bytes).

> *((Now you 'know'. Sort of... Now, please allow the following consideration:*
> <u>*‘Accepting’ and repeating means little, reduced to a deceptive social show.*</u>
> *Some call it education, some call it social interaction...It's just faking.*
> *It always has consequences. 40 years waiting is a local, minor one.))*

How to implement an 'impossible' compression?
Simply and short: Engineering!... Partly, a technique long used by CPUs, repeated by Woz. There are lots of patterns to explore, most allowing 3 bytes on 6 to be replaced by a single token (with an occasionally 3 to 1 gain)... *NOTE: It's prejudice what blocks natural solutions.*

ROM CharDefinitions reduction (8 to 7 bytes/char). By using a short code+table achieved **6.6** bytes (Freeing 25 Bytes). A later solution (demanding a bit more code) reduced the table to **4.6** Bytes. How?!? Patterns must be found, not learned... Then be worked with care to maximize reduction. Naturally, the 'trick' is to identify elementary functions. And a good balance. All natural.

A trigger example: 101.xx.yyy (as most Z80 opcodes) allow 4 routines and 3 bit arguments. Why a max of 4 routines? Each is an excess. *(Having time, we may get more opportunities.)* Not saying this was our final solution. This is just an example, very Zen on every sense.

Why this 40 years delay? *(And near 17 years not considering the mentioned 'request'.)*
Most people steal, build nothing. Some will copy/paste, even suggesting to be their own. We took 3 months, without experience on patching nor compressing. Work better in silence. At the end it was simple. When started we felt blind. Too many options made them seem walls. ***TIP!*** *Exercise finding things in the dark. Instead to wait for (believed) coordinates to them.*

A bit of unofficial Psychology, if we NOW may:
Notice most say to 'know', just believing? Mimicking what they follow? Look again. Don't stay there, question this: Having a reference, to copy and show, is...WHAT? Beliefs are NOTHING else than delusions accepted. Just conveniences sanctioned. ... Reminder*: Everything here is <u>shared</u> (ie, made available), <u>NOT ‘given'</u>.*

We share for each to ask (as mirror, his own character) what are the paintings each one do (without building) based on “everyone knows”, as on the more general 'show'. Unaware of a 40 years prison of their own (or 90 years). Proven by consequences. (Cryptic? Better be!) Now you 'know'... ((Learn to distinguish what IS, from what's merely supposed.))

As a lyric, cryptographic note (from 2017), here added:
A diseased world raises consequences. First it is overlooked. Later, it is hard to avoid :
- "A tragic world..." of thiefs and tyrants"... “A virtual reality of huge lies, ready to hate”.
- Most are ready to lose their sight, replaced by interpretation, feeling great (just smaller).

Seeing it, we share. Yet, should not give our walk. Only point directions, as enough (plenty).

___
 # We all speak another language: The one we build, with different reaches... Our own.

Patches and Changes

These are the results, from the ROM Patching adventure.
Exploring every system mapped, going were no ACE has gone before.
(Please read the above by sound of a well know space-opera, without wars)

ACE-Forth Benchmarks and ratios (respecting real MHz)
| -------------=== on 3.25Mhz ===------------ |=3.5Mhz=|

|**Real Ratios** vs BASIC (by WEIGHTS w/ Sieve equivalence|

	CPUstk	PRIV	**USER**	**TOOL**	ORIG	BASIC	
weights	121	219	236	259	275	4235	weights
xBASIC	35.0	**19.4**	**17.9**	**16.4**	**15.4**	1.0	xBASIC
xACE	2.27	1.25	1.16	1.06	1.00	0.06	xACE
/ASM	**14**	25	26	29	**31**	529	/ASM

|**Sieve Ratios** vs BASIC (natural FORTH: FILL is a Prim)|

	CPUstk	PRIV	**USER**	**TOOL**	ORIG	BASIC	
minutes	1.7	3.7	4.4	5.0	6.4	**79.6**	minutes
xBASIC	45.7	**21.7**	**18.1**	**16.0**	~~*12.5*~~	1.0	xBASIC
/ASM	12	26	32	36	~~46~~	615	/ASM

|"Previous Sieve (unnatural, FILL as a non FORTH word)|

minutes	*(na)*	4.1	4.9	5.4	6.8	*(same)*	minutes
xBASIC	*(na)*	19.3	16.4	14.7	~~*11.7*~~	*(same)*	xBASIC

Notes: (All) Proves ACE-ROM Sieve result was an artefact (Weights is immune)
Real Forth Weights3-Set clks-average (allows Sieve conversion)
TOOL Results represent next-day-ROM (+12 hours w/ testing)
values ACE and Spectrum Measured with single precise tool
CPUstk ACE2 “would-be" values (1 month lost, more gained)

Status:
TOOL Word **OVER** (restored) and **CODE** (recovered). Uses +25 bytes
USER Restored **OVER ROT** and **‘-’** (minus), in place of the CODE word
PRIV Optimised. Plus **+! 2* 2/ R@ 0 CODE** recovered (nothing was removed)

Purpose:
To show what ACE-Forth should had been... without external 'gentle' pushes
(We can still observe a disruption of FORTH. Casually disguised as support)

Description:
TOOL Is a ROM using 6x8 Tiles and optimized code to load them,
getting enough space to recover OVER and restore CODE.
USER Shows one-week-later ROM Sieve results by using only 25 bytes just as
TOOL results represent 2 days and 3 bytes. Both keep MAX and MIN (45 bytes)
PRIV Experimental patch to evaluate several aspects of a 8K ROM instead final 5K
VOCs It is possible to restore Parent-Child inheritance, using SCOPE instead PATHs
ACE2 Examining the (reversed) implementation of both stacks, against Z80 traditions

Compatibility: Public ROMs are designed to keep compatibility with previous ROMs, even snapshots
(aka emulator saved-states). TOOL and USER keep both ways 100% compatibility.
These are certainly closer to ACE-Forth than the burned ROM (!!!)

Licence: All patching works made available (as TOOL is) are under GPL3 licence.
*TOOL is available on our "JupiterACE" telegram account. USER *may* be made available.*
We had decided the CODE word importance to be greater than restoration of other words:
- Shared as a service to all those with an interest beyond "I own". It eases loading most.
We were wrong, thus the USER patch. People want numbers (numbers to check, not Roms to use).
So we now present both, showing what needs to be shown. More would be too much (it already is).

TAGLINE: Whom cannot... usually try to manipulate those who (with will)...can!
 # "A great pleasure in life is doing what people say we cannot do" ~ Walter Bagehot

Chapter X - No need to know

There's more to ACE-Forth than what meets to the eye.

The few people who examined the disassembly process knows this.
Because related with other innovations, as are DEFINER/COMPILER.

These are special, in their 2x2 matrix of Definer/Compiler * Ctime/Rtime.
Very special, intriguing... a puzzle hard to implement (harder because of SoftStack being used).

There's also the new Open, Scope-based Vocabularies. These would ease and simplify programming.
The traditional List-based system was obviously removed, as it would not fit 5K.
Even less expensive, the Scope based solution would needed at least 50byes more.
This left the small problem of "being actual" which we can deal with.

Finally, there's an opportunity lost that could be solved in about 3 days (we experienced it)
then improving global speed by 6%. Plus a few tweaks, if a few more days were available.

On the next pages we will focus on these last two subjects.
The first is too tricky to expose... hard and useless.

X.1 Search, and Vocs

The purpose of VOCs is to show or hide, so to manage what is to be found.
There were several options, namely:

```
* 1st was FLAT solution (scope defined by context):
  VOCS are closed entities. Allow a controlled search.
  Are located on the ROOT, adding single tree branches there.
  Context branch is searched before ROOT. Only those are searched.

* 2nd, LIST solution (a list of search paths, not the Forth instruction):
  Are located anywhere. VOCS need to be closed. ('de-facto' standard)
  Search list expresses as a stack (using an array or linked-list)
  Supported by the following control words:  ONLY  ALSO  PREVIOUS

* ACE's SCOPE solution (defined by context instead of locality):
  VOCs are Objects, allow heritage, and are open. (ACE choice)
  This MAY cause clashes, or undesirable search repetitions IFF
  not distinguished from traditional VOCs, or if replacing them.
```

ACE structures indicate an initial 2nd solution.
Forced to revert to a dubious one where VOCs are anywhere Open, due the removal of no VocStack.
But parent new data not seen, heritage is not fully 'auto'. That... is the glitch.

ACE VOCs replaced traditional VOCs (thus the SCOPE 3th solution). However, not completely.
An undesirable glitch is that scope is not complete (only previously defined words are seen).
Better would be closed VOCs, unless these are LIBs (CLASSes). It may have been original idea.

WHAT can be done?
We know VOC siblings hide data (access or block are part of its purpose).
Know with the removal of VOCS path management, scope tree searching remains.
After much study (examination, not reading) we see 3 chances. None exclusive.

Options by decreasing order of RomBytes demanded:
(1) recover the 'list' solution.'Usual' but too big for a small recovered space.   (~150?)
(2) patch ACE 'scope' to real Open-Vocs, loading from tape a small set utilities.   (~70)
(3) force the 'flat' Closed-Vocs, ie, imposing a two VOCs search: ROOT and CONTEXT. (~35?)

In short:
- Std PATH offers configurable Multi Vocs... Versatile but management demanding.
- Auto PATH force single path Open Vocs. Closer to Classes, working by invocation.

Meaning: ACE Vocs are not standard FORTH Vocs, they are closer to Static Object definitions.
Yet more adequate to excessively small computers (as the Jupiter), and a model for uCPU FORTHs.
... An unneeded reduction (saving pennies), leading to a very nice alternative

To be complete, a few changes are needed in search behaviour, dynamics being slightly different.
To be nearly as powerful as std VOCs, scope easiness must not become a source of difficulties.
Two NEW words are needed to manage Scope changes on compilation, while stacking CONTEXT.

We have 2 options:
- Restore of the usual Forth search system (closed Vocs), using the remaining of the linked-list (now slower due the loss of back link). ...Or...
- Keep single-path (open Voc), of smaller code, needing an also smaller set of support words. Still nearly as powerful as PATHs, are easier to use.

TagLine:
A known solution is an illusion usually copied... Being lost, is an opportunity of discovery.
It's educative to be lost instead trapped by the proverbial "I Know!".   This, a locked "way".

Vocs as a Tool

To manage development is also to avoid equivocations. Compartimentalization is needed.
Be it as directories (now called folders), be it in libraries Or hidden as in OOP,
the purpose is to reduce complexity by (first) reducing quantity.

Forth answer was/is Vocabularies. Priorities on search following Last-In-First-Out rule.
(Remind PATH on DOS: Tools away from sight, found when needed.)

HOW was Search implemented?
On standard Forth, VOCs are closed entities managed by a (PATH-like) list and related tools.
Initially ACE FORTH did not implement search paths as an Array, but as a list (still there).
This being more rational, the related tools are a bit bigger. Heritage is another solution.

Not fitting 5K, tools where removed. Or Floats would be lost... Tricky choice!
Heritage come to rescue, the mechanism differences being very curious and exposed nowhere.
As VOCs structure was reduced, a bit remained, enough to be used later. It can be restored.

Residuals remained.
The linked list remains also suggest a flagged link (3bytes) as part of the Vocabulary word.
Quite elegant! A node can be active, or absent (zero). The sequence? A list is a pointer.
Turned off, every node stayed on its initial state, with a zero flag, invisible.
Naturally, there would be a second link for efficient go back. With the removal of the tools,
that link (2 bytes) was removed. What remained (very wisely) is less efficient but enough.

ACE-Forth improvements... and 'cuts'
On the ACE, VLIST is no longer a VOCs list. But a words list trying to be 'open'.
We may guess ROOT, FORTH and a USER Vocabularies, joined as FORTH voc. Also DOUBLES, FLOATS and
maybe MATH (for SQRT, RANDOM, and trigonometric functions). ALL linked by the search list as was
discussed when FORTH-83 was being prepared ('things' do change) to avoid inconsistencies (a MUST).
Note: A faster RANDOM, by Marsaglia in the 90's, is now implemented on our Primaries Library.
It's the only correct Z80 implementation you may find (we checked, we had to build it).

"Almost got better"
Removed VOCs management, Auto-Scope was introduced, or may co-existed. There's a reason to it:
It's the basis of Procedure hierarchy, mother of OOP inheritage. We should also note that an ACE 'voc'
is more a Library, door to static CLASSES (FORTH was proto-OOP before Auto-Scope gave inheritance).
Auto-Scope is used by many implementations of OBJ FORTH. Willing or not, ACE was one of the first.

... **Not quite!**
There's a glitch in present Search mechanism: New words on a parent OBJ/VOC, stop to be
seen by the child OBJ/VOC). VOCS could be called OBJECTs, coexisting with VOCABULARIEs.
Precedence is enough to replace PATH-list, when we build a VOC as a child of another.

Management become it is difficult. Due the glitch, they must be used AS closed vocabularies.
To avoid inconsistencies, these should be used after the new FORTH vocabulary, and AFTER FORTH
standard words restored. As now, the VOC scope is limited to words defined BEFORE the new VOC

HOW can the search system be improveded? That's the tricky part.
- To build an access word for every Vocabulary, with similar name. Or,
- To change the Search system, VLIST a LIST again, no longer WORDS list.
  Meaning enforcing VOC closed scope to allow controlled PATHs. And spending
  a more code for doing that... Or leave it ready for a few loaded management words.
- To redo the system in a nearly compatible way, based on VOC precedence, forgetting the list
  initial system. Not depending on WORDs precedence across vocabularies (an ineffective auto-scope)
  then spend a less of bytes. (A compromise of funcionalities. Wider, less controlled scope)

Already checked, all those can be done (with a great work-time cost).

___
 # 'Impossible', can be a great teaser.

Got Space... What Now?

The problem
While OOP-similar was an unlikely intention, smaller Open Vocs would allow heritage by replacing Vocs search with usual Procedure Scope. This was partially implemented.

Because SCOPE is nearly as powerful as PATHs (also easier to use) it was an ACE sound choice. Not completed due time and ROM space (or would sacrificing more words, unacceptable). it would demand to move FLOATs to a library tape loaded library (acceptable but undesirable).

Even if incomplete, ACE VOCs serve *the* isolation purpose, while allowing needed core access. Allows 2 VOCs to be searched (no more), Context and Core (thus the advices on “clarification”).

A PATHs list, or a full SCOPE ?
A compromise must be found and implemented, between:

- To restore the more general VOCs (closed, plus paths)
- and Classes allowing heritage (without inconsistencies).

After a 2 weeks rest, to forget and then look again (unloading, thus more clearly) we decided ACE SCOPE method would be practical, after also considering the following:

- Std PATH offers Multi Vocs, but manual, context+libs (Forth-83)
- Auto PATH force Open Vocs, single path, context based (Forth-79)

Correcting the glitch is enough !
Scope was a sound choice for the ACE, in face of so little space available (with 5K ROM available, the ACE *is* a 3.5K FORTH (estimating ~0.5K decompiling support).

Auto single-path (open Voc) is smaller, SCOPE is nearly as powerful as PATHs. And easier to use. Will demand smaller internal changes for support words, still leaving us some space.
... Dropping everything, can be a good thing.

What do we have ?
We have chosen a limit of ~50 bytes of added code. Patching code still demands ~75 bytes. Support words (while demanding an extra 36 bytes on ROM) can be loaded when needed. Making (for now) a total ~110 bytes to be used on ROM, reducing Words recovered.

For easiness of use, and only 2 support words, we remind C.Moore mentioning, we quote, “VOCs may have been a mistake”. Not by themselves as closed Libraries. But by the lack of SCOPE search dynamics. These allowing heritage, more adequate for 8bit small systems, even self-sufficient/programmable uController Systems (as the Jupiter could have been)

((An actual example would be uControllers carrying their own compiler inside, no longer fixed and Host programmed. --- A special mention: The Fignition system board))

SCOPE is not ANSI-Forth, but is a solution very much in the spirit of FORTH.
It's solved, designed, not implemented. (Why would it be? Maybe it's too late!)
Anyway... Dropping everything is usually a good thing. A way to start fresh, anew.

P.S. More adequate to experimentation:
An ACE clone on more convenient hardware (of unrelated code CP/M directed), for Z80 DIY computers. Or available FORTHs on DOS, as the small PIGMY. Or the very complete F-PC (that replaced F83). These FORTHs mentioned, replace the ACE on better Machines. But do not give the same easy start.

For serious, actual and Multi-OS, there are SWIFT (by FORTH inc) and GFORTH (free GNU!).

___
TagLine: A known solution (usually copied) is an illusion. While "to feel lost" can our opportunity for discovery, it's educative "to be lost" (in a field), knowing a solution to be... just a line.

What is in the Public Patch

Visible differences... (There's a bit more than seen or here told)
To check, run the SIEVE on the ACE (both ORIG and TOOL Roms). May check the Spectrum too.
(An ACE snapshot is available to check both ROMs, quickly and easily. Another to the Spectrum)

Spectrum only: PRIMES program should be limited to **1x** run ONLY, not to suffer 1 hour 18 min wait!

Additionally, now just for the ACE (whatever ROM), anyone can try the arithmetic stress test from PCW Magazine adapted to FORTH. Must warn it to be a very, very naive test: Result is mostly DIV:

- '/' (div) improvements: On obsolete ACE+CODE was 18% ... **ACE_TOOL** by **23%** ... ACE_USER by 34%
  (PRIV patch only improved div by 39%. It shows how appropriate the tweaks on 25 bytes were)

Changes:

- Speed changes are significant. (Around 10% are perceptible, above 20% become very obvious.)
- On a patch, CODE word was added (the only change listed). It's a non expressed invitation to either implement or restore primary words. Another reason is the example shown on the manual is NOT a definition of 'Code', but an example of 'DEFINER' usage.(needed one way or another).

The small downside of +CODE (emulator snapshots not compatible) has been removed: Now replaced by the ACE-TOOL Rom. (We restored the RAM-ROM link previously 'removed' for the important CODE word)
*We felt hard to drop **+!** from the first patched ROM. Considering **CODE** builds all, and the need to SHOW the "why" of Sieve failure... We considered it was a sound choice. We still do !*
So, and at the moment, only ACE-TOOL was made public: It serves its purpose well.

There's more to it:
Those versions are based on our 6 lines Char definitions code (with 25 bytes made available).
The ACE-TOOL Rom is fully compatible with snapshots, BOTH WAYS. Same thing with the ACE-USER Rom, a faster variation of the same tech (art is not a copy) but without the CODE word.

Why the CODE word?
Reason for the added CODE word is to BUILD Primaries. It only builds, it does not run later.
With CODE becomes 'natural' to load missing primaries. It fits well on the original ROM...
Either for use, either for development if any. (For the moment, it serves its purpose.)

Another reason for the CODE word:
The example on the manual is only an example of DEFINER. So it carries a small delay cost.
DEFINER CODE example 'may' have preceded the removal of the CODE word from ROM and Manual.
Using the restored CODE word creates 'real' Primary Words, ie, with no such small delays,
To/From TAPE effects: None. ACE FORTH was ready for 'code' words, as for 'defined' words.

There's more to the Patch than the eye can see. Side improvements? Only speed can tell.
Other patches (those breaking the 20x barrier) are not really needed. For a reason:
The Primary Words (restored on patches) can also be loaded... With close results

Who would ask for 'more'? (Later we found a chance to a cleaner Ace-Forth, the USER patch).

And a reason NOT TO place CODE on ROM:
The CODE word can be replaced by ROT on a patch variation, still using our 25 bytes solution.
As the change is trivial, it sounds possible (not enough) to make that one simple replacement.
((Even someone to say all to be his/her work (quite false). Just an edit of a previous work. 'Shared' deserves respect. Preventing abuse, our USER patch is a bit more than a mere replacement.

Any deception can be quickly detected, as it is not enough to fake the USER Rom.
Everything is sharp on USER. Another alternative would be the compressed system.
Previously done, it is private. Thus allowing experimentation of other options.

Will they (or 'it') ever be published?!? ...Why not? Yet, what for?
While these questions wait, TOOL is good enough for all purposes.

X.2 Another Exercise

The ORIG-inal Rom, shows its evolution.
It's a lot as knowing how a cloud behaves. Then mention what will come next, and someone not seeing it will say “Prove it, show it to me” ...while actually meanning: “Make me see”. Such request, seldom a command, is useless in the absence of the main supporting tool.

An example:
The cloud vanishes from sight, still there. We know it is there, but there's nothing to ‘see’.
When the demand is for an accept and a brag, the demanded proof is not understanding it at all.
Is just a mean to say “I know”, being an “I accept”. Even 'knowing' is not understanding.
As a matter of fact, “communication” notion fails: ‘Information’ becomes petty talk.

Lesson learned:
Thus, as when reminding speed can change, it was not listened. Yet, there are different speeds.
At least 3: Architectural, Available, (initially) Delivered. But most people what one number.
We have noticed, repeatedly, the motivations behind. We have confess it was unexpected.

Other lessons observed:
And, because unexpected, a lot of nasty things happen later. To hide it, or as revenge for one's blindness, as if evidence was offensive. Not wanting to change, most react that way after creating false images of anyone they suppose to be familiar with.
A Davidean complex (as I call it) may be revealed, can be quite destructive.

Making a patch or many, was the only way to prove the evidences.
However, there are other subjects also discarded, beyond the subject of speed.
Speed was a terrain for equivocations and lies. It needed to be placed on a Rom, as is.

That other subject, is the evolution of the Firmware. Again, the purpose is educative:
To show how limited ‘official’ statements are (only serving to fool one's neighbour).
Again, we can not ‘show' what is visible but not seen. We can only build what we see, then show it working. Or in this case, because not a gift, just transmit the results of the 1 to 3 days later Rom. We know the results (~19xBasic)

The Final Rom That Never Was (that would, suggested by the Listing, in 1..3 days)
- Is faster than our USER patch, near ~19x Basic, both Weights and Sieve.
- It's almost obvious, much easier to implement than our hack patches.
- All hard work done, it was "around the corner" (we guess 2 days).
- Even better... it does not need our compression solution.

Contrarily to our patches, its code generation is quite different. It results from using Primaries were Primaries are due, and allowing RSTs to waste a few bytes (also were due). Improvement is better than our restorations. And this without using CharsTable reduction.

Again, we observed what we are saying *(repeatedly proven, even making it easy to check)*
We will give no extra information to whom has never done nothing beyond objections.
We learned not to feed the inevitable destruction of what goes not understood.

Why saying it now? Such Rom is now ‘incompatible’ though the real thing.
We did forgot the whole subject, distracted by ‘fans’ lack of interest.
Our focus was centered on correcting lies, upgrade everyone's sight.

So we now remind:
The Jupiter ACE would EASILY have a Sieve of ~19x, instead 12.5x (actualy 15.4x). So easy it is, to build it, once seen, a pair of days could be enough. Why should it happen now? We ask.
There was more to the ACE, than just speed alone. 40 years later, it's still educative:
- The Jupiter ACE, to me, has been a lesson on the World... One not to overlook.

___
 # Lesson learned: Not to offer pearls to whom would disolve them in vinagre.

A LATE rebirth, and Compatibilities

All mentioned on the previous page can be done with enough ingenuity and determination.
Even without grasping system details. Yes, it's possible. We will remind how:
Just do not be the guy mentioned in “Let him do our work for us”

What had happened, by late 1981?
It makes sense, on hurry to meet a deadline, for a systematic drop of words (to fit on 5K).
One by one, keeping the essential. Turning a few (less used, as OVER) into Secondary words.
We say hurry, because the evidence of previous reductions in size of many words (as '**-**')
in a 1st attempt to reduce losses. Then come the observed systematic trim of words.

MIN and MAX were placed a bit away, when 5K was reached (without time to remove them, restore other).
This is another reason to assume an hurry. The cut being incomplete, not allowing to get a balance
and restore a few losses from systematic cut. It also suggests management lack of understanding.
By our own experience, rigid demands can be harsh (specially when hiding personal flaws).

What results from not having ~3 days? *(our evaluation, knowing each independent Word size)*
Mainly, Primaries turned Secondaries. Not having a clean Push/Pull RSTs (12% faster, cumulative).
  * First, note the 'Late' ~3 days Rom is internally different, not compatible with the official
In spite of its advantages, namely a few extra words, "after the fact" is just a curiosity.
It should not, but the machine is dead. Even if the firmware still ticks and shows "other ways".
  * Another option, to faithfully represent it, is to make it <u>mostly</u> compatible.
This can be achieved with extra work and our 25 extra bytes free hack. It would raise objections
on faithfulness after mentioning “almost was”. The added compatibility using our +25 Bytes hack.

ORIG Rom lost mainly speed and ‘+!’ word. Not a big deal! (see Roms Comparisons PNG)
T<u>he essential was delivered: A structured language</u> capable of to rebuild, define the words lost.
Still better and faster than anything else available (the programming, not the flawed hardware).
(Check all the 1982 goals... Are well reminded on the "The Jupiter ACE has 30 years" article.)

"After the fact" corrections can be problematic (and this not only by themselves, as we know.)
People who do no longer use the "little beast", just “own one”, would complain “not compatible”.
Would also complain on its alternative, because compatible (even if without optimizations added).

? What would the changes be ? It's a matter of “code balance”... *(It's also a matter of “diving in")*
It only needed an ending of the systematic size reduction... Then using a leftover: MAX and MIN = 42

Here are the priorities:
1) One or two dropped words (By importance, ‘+!’ is #22, ‘2*'and ‘2/’ are #30 bytes. **-** 22
   With time to evaluate (importance/size/speed), choice is '**+!**' (priorities do help).
2) Push/Pull RST's as they would if 4 bytes were available (an apparent loss of 10 bytes).
   Really, an overall loss of 4 bytes (now available to be lost). **-** 4
   Speed benefits: 13% on every Push or Pull (altering global speed by around 6%).

After this point, things are not as clear (there are many combinations, thus 1-3 days) = 38

3) Stack Operators (OVER ROT) back to their original state, as Primary words they are. **-** 7
4) And/Or restoring hidden words to primaries. What can be found? What to chose? (= 9)
   Found we could also restore 2* (it does not affect Sieve), with 3+2 days to complete it.

? Can our +25 Bytes Hack be used ?
Yes. To also get "internal states" compatibility. Yet, further optimizations should NOT BE DONE.
This, respecting the historical goal. Questions are: Why do all that, to whom? What uses 8 bits?
Now that would be a useless task. *(Worse, it could attract the attentions of many smiling wasps.)*
(Note: Rebuilding the LATE Rom, we left Priority-1 to last: ACE32 emulator prohibits RST-5 changes!)

Our conclusion:
Compatibility is NOT a goal, understanding is. (Hard work deserves a fair use! Does this makes sense?)

___
*# Little things, determine other big things... ~William T.C. Fort*h

-

Appendix

A.0 About this Project

What is?

ACE ROM code.mesmerized us by 1983 (in spite of running on 1k).
This started with an attempt to organize ACE code. To understand it (and FORTH).
Documenting ASSEMBLER code was initiated, replaced by the recovered listing.now available.

The documentation goals where expressed as 3 needs to satisfy:
- Precise   - “Must be correct”
- Clean   - “Should be readable”
- Descriptive - "Will try to be complete”

All the above rules would apply. To achieve those purposes, the K.I.S.S. rule guide us.
Not limited to the disassembly we used extensively, we learned much (Thanks to the literature referenced in the "Special Thanks" page. So we achieved (never final):
- a global view on the code, and
- some code tweaks (not as easy as desired) were (and when) needed, with
- A grasp to a possible PC clone (clone is an excessive word, to honour is closer)
  I.E. offering benefits exclusive of ACE-Forth, never repeated. That window is closed.

Goals:

Understanding!
Knowledge of every aspects of this particular Forth implementation. Done.

Readability!
Clone Size is NOT imperative as was to the original. But readability should be.
The Z80 clone design was a delightful experience Both ACE2 (initiated), and ACE++ (completed). As stated, there's no reason to implement it. The hardware simply does not deserve it, nor such effort is justified.
The creation of new code must not disturb the documentation of the original ROM.
The patches kept user codding just the same. Are fully compatible.
A clone is another matter, an interesting exercise be it of compatibility of either code on tape, either of snapshots. Beside the exercise, we must confess to be totally irrelevant (anachronic).
The creation of a clone would be more useful. But there's no benefits, nor returns.

Complete!
Allow to compile a clone (sort of) and/or changes of the real ACE.(done).
Different CPUs could be used, benefit for the language itself (ACE-Forth).
For this purpose, reorganizing the code and cleaning would be considered, at the expense of some words or libraries. (the impossible compression of the Chars-Definition-Table offered a chance without losing the Floats Library)
For the purpose of compatibility, Forth Words addresses should be kept the same.
This was a splendid decision. It allowed the recovery of ACE-Forth to be closer to its intended speed (TOOL and USER patches). Then with more bytes available, to restore some words not fitting on ROM (final composition to ‘decided’ later).

A.1 Project versions

INITIAL GOALS:
Amazed with the versatility of ACE-FORTH, we never forgot it. We the wanted to:
- Understand ACE FORTH, also the Jupiter Hardware. Have a better grip on FORTH.
- Decipher the ACE-ROM. Make educated decisions for the future.

RELEASES:
v1.x - Clarify and Comment (IP on stack hardens sight of some words. Also slows the whole.)

Organize;	*Done*	Kernel;	*Done*
Forth words;	*Done*	Compiler;	Mostly Done.
Routines;	Partly Done. Some 'special' Words need re-examination.		
System Vocabulary;	~Incomplete, as some parts were removed from ROM~		

Optionally:
Code Changing (as soon as feasible) to:
? Pseudo-Code translation; x86 translation;
? ACE-PC: CGA mode on CGA,EGA,VGA; (ACE compatible)
? ACE-Plus: COLOR and +128 redefinable chars; (CGA+VGA native)
Options above are either Delayed or Abandoned:
((Intent was a system for Education, on Schools and University.
((Preliminary work 'lost' by 2009. Also, there no 'official' competence.
((So it was (is?) a useless task --- There are other FORTHs... Use them!

v2.x - Not published (only 1 initial copy was privately shared, without reply)
- Added original listing reconstruction (a work in progress)
- Many missing words where made available. Published, received no feedback.
  Apparently, there is no interest. So, wait for ACE 30's. (this was circa 2012)

Fact was, is, the Jupiter ACE gangs seem more interested on keeping secrets and a status-quo.
Keeping everything 'controlled', even fans 'controlled'. Personally, I abhor 'controllers'!
Not helping while trying to look so. Do 'get' free contributions, to own and get more.

Yes, to "fill needs" allows much, as a tool distorting more than OVER not primary.
As smiling dark clouds, grab much to share little. "Do this, get us that".
The "Give me" culture seduces on an attempt to manipulate and 'own'.

As a result of that sad reality and discomfort: *(# Creativeness is a lonely task)*

v3.0 - Private work (Notes&code = One thousand pages) ...
- Restored original listing. A welcome replacement. Added, as both are available.
- Added Changes to Patch ROM. Already designed and ready to implement.
  (It turned "To publish or not to publish?!?" Then to distribute selected patched ROMs.)
- Again, maybe to wait for the ACE 40's, by 2022. (Near there, we may not reach it.)

v3.x - Public version! Now using Telegram and Google drive (due easier access). The ACE is ~39y old.
- Changed sub-version numbering from v3.n to "v3.<day>"
- Added my notes on the inner workings FORTH. Educative.
  (Less equivocal details than the available. More of its Essence)
- Corrected language inconsistencies caused by years of changes for private use.
  Mostly result of editing in English (a foreign language, thus an extra worry).
- Progressively better organised, more readable, never perfect. Hopefully useful.
- Completed by December of 2021, after too many 'changes' and accidental losses.

Note: v3.x shares a listing, restored after the listing sent to a German firm.
It was present on the undisclosed ACE ROM Doc. Proj v.2. It's now revised, on Book 2.
Also means private comments and code are kept private, to share when found adequate.

v4.x - Is a revised 2022 v3 text, with a few additions. It's numbered "v4_<yy.yearday>"
The ACE will be ~40 ! We want to end our small little PDF cathedral, by then.

A.2 Some Thoughts

In general:

- This was a nice system. Not for its hardware, but for its firmware.
  Effort to build an useful firmware largely compensates the deceiving hardware.
  Even the firmware was reduced to a minimum. While with an inadequate stack, it was inspiring.
  ... It still is! (a true 'ace': For the innovations added, ease of use... and as a memory.)

- Only with a pair or two of inexpensive chips, better results would be achieved.
  And a relatively expensive Chip (the MUX) was not needed, paying the other.
  It's understandable that options were too many and confusing, thus tempting to
  fall back to a 101 'design' with a Video section added (the ZX-80 should had been).
  Colour was a 'secret' quietly shared between some academics, the most elaborate were implemented on Chip (as Commodore did) or transposed to an ULA (as happened with the Spectrum, not to give profits to Commodore). Chip makers later followed.
  Its simplicity is STILL obfuscated with TV data theory (a colour is just a delay).

- It was surprising to find at magazines, some with a due enthusiasm with the FORTH offer
  (showing its advantages). Another, unforgivable: Pushing extremely incorrect information.
  - *We remind a "choosing what to buy" booklet, informing the Jupiter to run at a very slow 1MHz, with an 'obscure' (unknown) language. (BASIC listings are known by clarity)*
  - *We remind a ZX-clone named Ace, offered in the US when the Jupiter ACE was preparing*
  - *to be commercialised. Nasty... (obscure more likely, maybe more adequate)*

- It was even more surprising to see on Wikipedia, objections to the real ACE speed,
  even some occasional attempts to impose undue references to a competitor. Then
  drowning the entry with irrelevant 'information'. Though undesirable and common,
  are understandable. Forced mentions to the unrelated, is not. With one exception:

- Jupiter hardware is extremely similar to what the ZX-80 SHOULD had been without the nefarious
  insistence of using a Z80 Cpu as Video Manager. A trashy, damaging action, behind a legend!
  We suppose such idiocy (later corrected on the Spectrum) was directed by someone as arrogant as ignorant (unnecessarily difficulting the development of the ZX-80, and the later ZX-81)... While the non-able want to be applauded by luckyness.
  It's common: Knowing little can be worse than knowing nothing. If arrogant, into blindness.
  Then, it does not search, it tries to drive. Only the humble never stop searching and trying.

- What the ZX-80 should, is found as Jupiter hardware. Very similar, reason for some questions.
  <u>Not to question</u> Branding and contracts, these being a way to 'own' by legal fictions, either
  'owning' public shared Know-How, either 'owning' paid engineers. Even their private findings.
  All stolen (as patents now do) under the excuse of a contract and a salary...
  We see it on law by 'interpretation', away from the 'spirit' of the law.
  <u>But to question</u> keeping flaws. Keeping a Computer stuck as a mere CPU application:
  - ROM overlapping CPU vectors, when CP/M pointed the correct way (with no cost added).
  - Flexible Computer vs Fixed Application hardware

- VSynch being a vector, instead routine evidences development was made on CP/M.
  Another is its position, though that is just an hint. One on many.the code exhibits.
  Repositions of code and resulting 'jumps' increase the suspicion of a previous clean code
  developed on another system. At the time the professional system was CP/M, all tools available
  did run under it. Thus (and also) a need to reassemble with reposition, over previous listings.

- Had the Jupiter hardware followed the CP/M design, as it SHOULD:
  FORTH would be in place of BDOS and BIOS (system routines and tables) on a separate 4K.
  All at the end of the 64K space, Video space too, would ease development and allow 'changes'.
  This also would an extra ROM, if needed. Not needing the MUX we all can see, suggesting
  no real work had been done beyond mounting an old proposal on a breadboard. Understandable.
  Just not acceptable! What happened? And WHO was the secret investor, hidden owner of JC ?!?

Questions never asked

The great question one may ask:
 - Why not build a good Jupiter, to the praised ACE-Forth ?!?
There are several answers, but their common end is:
 - *"No one wants to use that, not really."*

Its Time is gone! It was not built when due, no reason now.
ACE-FORTH firmware has shown (by then) how a Future could had been.
Only a few cheap changes to the inherited Jupiter schematics were needed.
We ask: Was investment on the Jupiter ACE, an evaluation or a kill? Or even both?

An old proposal Sir Silly refused, imposing his bright ego.
Let's remind what was never told by the narrative sold:
 - A fascination with using the relative inexpensive Z80 to do <u>all</u> the work. Result was the ZX80/81 only used **1/3** (or 1/5?) of the RELATIVELY EXPENSIVE CPU. That <u>3x inefficiency</u> is evidenced by the Spec/ZX81 Print a char ratio AND optimised code (so 1/5 => 1/3 ratio).

The imposed non-sense created difficulties. The ZX-80 <u>was</u> an ordered hack, harder to built.
<u>Also</u>, it demanded extra circuits to support it. Those extras could had been dedicated Video (those we see in the Jupiter). Beyond that, we see no architectural improvements on the Jupiter.
We see a 101 schematic + that Video section (most of the board)... We do see an opportunity lost.

Beware: 'Leaders', 'Geniuses' and 'Heroes' are rarely real
((Note: We've all seen 'management' trashing everything. I've experienced that.
I've worked under orders of a complete fool, Eng. by title not by merit.
Directing with Magazines 'knowledge', and demands "for yesterday".))

This was not the usual 'scenario' experts of lurking are used too. But was/is the reality of how the 'system' works behind 'show-of' and 'talk'... Accepted, hidden in plain sight.
Under "the narrative" (enforcing virtual realities). The world has been trained to accept blindly any arguments by a'look' ... Technical (or otherwise), gossip (or otherwise).

Difficulties are never 'the' problem,
but directing egos installed in management are. With orders for engineers to solve their delusions.
Mostly, deciding on matters they do not understand. Mostly managing hypes by the flip of a coin.
Forged leaders do smile, are even close. Being a source of deception, waste, even danger.

BTW: Woz was very lucky... not to be stolen by Jobs. We give Jobs that credit, because rare.
Do not accept the tags applied, those belong to Woz. Call it an opinion, it's just observing.
((NOTE: We suppose management may had been the reason why ACE-FORTH was not rewritten with the recent solution for the Z80: Would loose 1 month. Would ease development, would gain 2 months. Would double its speed, reason why FIG v1.1 speeds are shown.))

Concluding on the Jupiter-Ace tragedy of brilliance and (...):
Now there's <u>NO reason</u> to build a non flawed Jupiter. Its Hardware looks a too late correction from (Sir Silly) idiotic impositions . We fail to see real work (would make a difference, for pennies).
Also, it is <u>not practical</u> to build ACE-Forth on new hardware. It would had one user.
Instead, there are many good FORTH's around, all available (and with disk access).

<u>An opportunity was lost</u>, not needing descriptions of 'weirdness' (nor being a ZX-81 correction).
Nor a Magazine booklet on "what to choose", advising against 1MHz Jupiter ACE, its slowness.
We have seen promotions to the Jupiter, due Forth. And attacks saying Forth to be 'obscure', a characteristic belonging to BASIC after it's batch nature (reason for extreme slowness).
We have seen the 'BASIC' name applied to C and Pascal compilers (as a marketing naming).
As the public "must be kept deluded". Where to? ...An enslaved "Brave New World"?

___
 # Only reason to correct the ROM, is thankfulness. Will you use one ?!?
Maybe not, nor the point. Maybe you will, now for a few minutes. Again, not the point.

A.3 Other CPUs

Why TO PORT the ACE *(An optimistic view.)*
- The Z80 has now better replacements.
- ACE's FORTH is uniquely appealing.
- The hardware was self limiting.
- Direct and SBR are faster

Ported, would have more use than any emulation. Main question is, what use is there today. Still, it would be nice to see it still present, even to evolve on different CPUs and hardware, either on the PC or on uControllers. Then, why to invest time and effort for an “I have one” ?!?

Really, there are already some excellent FORTHs. So it may never happen.
The ACE is still is an excellent source as inspiration... Of what?!?
- Of easy usage, of 'redefining' and even easy compiler words.

More important than implementing, is the joy of the design problems encountered.
To recognize and solve those difficulties, after the initial perplexity.
That... is much more rewarding.than just saying “here it is”.

Why <u>NOT</u> to port the ACE *(The pessimistic counterpart.)*
* Though didactic, most people are focused on hosted C programming, following a trend of limitation and dependency (some call security).

* CPU makers follow follow and feed that trend, replacing BIOS inner code with host libraries they control. People accept that. Rarely knowing what's beneath, nor desiring what would restore their ingenuity back. Just “to mount”.
- Most people now only wish to brag or applause. Then respecting only "pay and obey".

___ ___ ___

Revivalistic Options:
The previous paragraph may be depressingly real. Still:
As a personal nostalgia, or a personal tool, it can have some sense.
Options abound, as long as kept private (sharing exposes us to strange reactions).

Then: ACE-PC versions, preferably native, can be used on several Operating Systems under DOSbox. DOSbox does an excelent job in x86 CPUs (identical instructions means no real emulation is needed) and a very good job with different CPU's to were DOSbox was ported. (IFF there was an interest)

Further, as a DOS program, it allows the programmer to recreate what the ACE COULD have been from its beginning. And also what it could had become on its future versions (the lost).

FreeDOS is available, and an option. But it demands source code. Once upon a time, open source seemed great. Reality says otherwise: Returns go to those more interested on to grab control, than to build (or to share). Wise rules ARE: “Never say everything”, “Share, do not give”.

Fact is old DOS (Quick and Dirty OS, bought by “the usual culprit”) offered a chance to use ACE-Forth as a machine-independent programming-language. ACE-Forth by the 80's, was useful for learning, exploration, control and recreation. Today it is wiser to use other FORTHs.
... They also open one's mind. (Special mention: 4CMP native compiler, by Tom Almy.)

Question is: Who use them? Only the dying breed of those building efficient applications.
... With faster CPUs, its a vanishing expertise no longer available to the commoner.

Back to the subject: Beyond the Z80, there ARE some possibilities (pure exercises).
We examine them in the next pages, under the cloud of (present) dark realities.

Away from the Z80

uControllers Option:
uControllers are very accessible, powerful.
Not built for interactive direct programming, they can.

Interesting uControllers for such a port are:

Parallax Propeller
- Extremely interesting CPU, potentially a System on a Chip
- Has two interesting FORTHs... (No ACE, not really)

Arduino family
- Quite popular and reasonably capable.
- Video should be considered as an external, auxiliary system.

Reasons not to port to uCPUs
- Not ACE-Forth, there are many. The Multi-task ANSI “FlashForth”, or the FIG "Fignition".
- It's common some ‘entrepreneurs’ to be rewarded by other people's true works.
- Returns are on Hardware, ie, Firmware efforts are quickly ignored.
- Most people do not respect work (even accept abuse easily).
- Applause is not a valid reason. Usefulness is.
- There's little use beyond a short try.

ACE-FORTH on PC makes more sense. A few directions:

Nature:	Either Emulator or Native
Compatibility:	*.com + 16 bits + BIOS(Disk) + CGA(320x200) + Beeper
Programming:	*.exe + 32 bits + xDOS(File) + VGA(ANYxANY) + SoundBoard

Can be used as a support, out of common usage, allowing user defined routines on programs.
Then an interface for programming, a script language added to programs one might build.
Though not the ACE, the utilitarian nature of the ACE could serve as an inspiration.

There are many Forth's available. Facing most, ACE's main appeal would be on education.
As an easy Forth, the ACE ghost is bright enough to invite many further usages.
However, the reasons not to port to uControllers also apply on the PC.
Regardless of such realities, we had examined what it could be:

I/O Changes
(I/O design changes needed... beyond VOCs boot-loading choices)

Display

Use an external file to save the VGA CharSet or have our own.
Compatibility must be kept with COM solutions. (Can CGA be an option?)

Keyboard

System or Translation Tables dependent? (BIOS is limited, GUI hosts are complex)

Beeper

COM -> Beeper
EXE -> SoundBoard if Present, Beeper if not.

Tape

COM -> Tape routines must be changed to diskette access.
(While possible, compatibility with ACE-CompactFlash Project is useless)
EXE -> Full DOS File access.

ACE-49k.com

In the 80's, an ACE-Forth on a PC would be a luxury, as much as the Jupiter was the opposite.
    Disassembly is no longer needed, using Blocks on diskettes, and files on Hard-Disks.
    However, its so convenient that it justifies (to be universal) a tiny drop in speed.

```
ACE-49K.com     -- 16 bits code, 64K limits do apply
    Coding: Direct mode is faster, but with a 48+1K space. Indirect mode allows 64k+64k
            ACE-Forth can be compatible with the ACE, even simulate its I/O

Compatibility Exceptions:
    CPU:    Use 86x         ; Asm code must be translated
    RAM:    Using Segments, there are several options:  CS=DS (direct mode) or
                (indirect mode) ROM in CS,  64k User space in DS.
                In both cases, Multiple Stacks share SS: 64K

    Devices: Buffers on ES:
    Video:   MDA, CGA, VGA   ; Controlled by FORTH Words
    Sound:   Use Beeper      ; Mono-task or independent

Enhancements:
    Color can be available (several options).
        On CGA/EGA, limited to the CGA 8 colors.
        On VGA, 16 colors, using palletes (KISS rule)

Compatibility: (As Emulator)
        Use the 2K at $3000 (up to $37FF) for ROM extensions.
        Use the 1K under User-RAM ($3800) as Buffers (2x512bytes).

Disk:   Use BIOS direct-disk access so it can be compatible with ACE projects
            so it can be loaded from diskette without DOS (direct as the ACE).
        Also use DOS file access so in can also be used under DOS.


ACE-386.exe     -- 32 bits code, user space is unlimited
    Coding: Direct mode is possible. Indirect mode is again advisable
            To use same Assembler is possible. The A86 has a registered pair: A386 Assembler

Compatibility Exceptions:
    CPU:    386     ; Some ASM code must be translated
    RAM:    Flat    ; Can be both fixed, or dynamically increased.
    DATA:           ; 32 bit words.

    Video:  VGA     ; Controled by Forth Words
    Screen: !?!     ; Any resolution (Text and/or Graphics)
    Sound:  Beeper  ; SBlaster can also be used, instead

    FORTH:  32bits  ; Big change! Still ACE-Forth? Color-Forth?

Enhancements:
    VGA Color& Char redefinition    ; Color Graphics Mode 13h = 640x200) ?!?


Notes           -- Beyond the uselessness of such a project:
No BIOS/DOS duality: Intercept Boot as a Jupiter ACE CP/M should. But 16/32bits.
Moving to an EXE file: Possible, but much more tricky. A ".COM" is a better exercise.
Another chance: Apply some Color-Forth Moore's ideas. Adjustments are acceptable, not more.
Kernel may allow ACE-Forth and Color-Forth Vocabularies. Can it be done? It's soon to tell.
```

ACE-49K Notes

```
ACE-49.com Purpose:
  The main purpose is to achieve MAXIMUM compatibility with the original:
    Compatibility is achieved with Words and SysVariables positions.
    Expansions can be available as options.

Targets:
  8086 CPU with
    Hercules and/or CGA, VGA
    Floppy (360 and/or 720)

Examples:
    EuroPC; Amstrad-PC512; 486-VGA;

===

ACE-49K.com Structure:
    1 user segment (64K space like the ACE) or
    2 separate segments (ROM and RAM) for ROM safety.

Screen:
      MDA HI-TEXT => 32 columns x 24 lines (B/W)
      CGA 320x200 => 32 columns x 24 lines (B/W + 8 colors)
      VGA 640x200 => 32 columns x 24 lines (B/W + 16 colors)
    Check faster VGA char definitions in CGA compatibility mode
    MDA, CGA and VGA Video as modules added.
    Full compatibility in FORTH only programs.

===

ACE-640k variant: File access should be similar to ANS-FORTH
    Also Boot from Diskette with the full PC available.
    BIOS/DOS duality is tricky but it can be done cleanly.

ACE-640k structure
    4 Segments (Code; Data; Stack; Extra)
    Full compatibility achieved on CS:
        No direct access allowed to ROM, on CS:
        No direct access allowed to Screen, on ES:
        Direct access allowed only to used space, on DS:

    ScreenModes:  use of 40x25 (as well as 80x25)
    DEFINER variables could/should reside in ES:
        A Turbo-Pascal HEAP solution to access 1 full MB
        The Programmer is responsible for such usage.
        Use of VGA controlled by Forth Words.
        Seems a natural progression up to the ACE-386.

===

There are a lot of good Forth's out there. There is no reason now to build a new one.
Designing one is fun enough. The exercise of discovery is personal, a very private walk.
As sky-diving for pleasure, not for a show. Or as being alone on a mountain, as we should.
What we are saying, apparently off-topic, is this: We need to be alone to really find anything.

To implement such designs is useless, on a world of quick applause (envies, and gossip).
Useful, at least to prepare students of uControllers, the tendency is to be Libraries dependent.
Such is the practice of most CPU makers, on a world that prefer to praise financial 'suckcess’.
```

A.4 The taken

For study purposes, the final production listing, reconstructed after the one 'demanded'
by a Sinclair Printers German builder. That turned useful: We can consult a few listings.

The Fair 'English' Disassembly (2002) (replaced)
Its origins are a mystery. Because an honest and competent, but anonymous work.
A true disassembly, hard work from an unknown. Altered for clarity, has sourced this
ROM documentation. It is now replaced by the Restored Listing (finalising our examination).

The Overlooked 'German' Listing (1990) (replaced)
Playing hide&seek for decades, it is no mystery: It evidences to be a translation of the requested
full code made by a German Printers firm. Very strange 'demand' (only ECHO details would be needed).
(What is the sense on... a dentist to demand your daughter to fully undress, to fix a tooth cavity?)

This listing makes available much valuable information (we are grateful to its *Penthouse* qualities).
Has become as helpful as unexpected: A descriptive mirror *(missing macros may be similar to ours)*.

The Restored 'Original' Listing (2020) (date on listing, 9dec1981, may mean "after request")
It is based on both the above mentioned, also on knowledge shared on the publications mentioned
in the "Special Thanks" section. Then ending with a silent work of guessing translation losses.
Many years ago we sent a version to someone. Then got a first readable text, for a rework.
(We are responsible by any glitches that may have remained.)

Those 3 listings mentioned above...
They ALL relate with the ROM burned version (ACE1, ACE-ROM, ACE-ORIG) 'burned' for the Jupiter.
These are NOT the extended developing code NOR the more complete words-set (we call "THE ghost")
confidently assuming it as developed under CP/M. Then painfully MOVED to this Non-CP/M.

We were very thankful for the reduced, 'demanded' listing that survived. It allowed much.

A personal view
We expect several work files (at least 5, or 7) to be the developing version source (the ghost).
And of 2 previous versions, the last to fit in such small space. Note: As time passes, we loose
files when accumulating, then loose the disks because untouched. We are glad this code survived.

What about the developing files, preceding the burned Rom?
While we would like to see it, we no longer need it. We consider to be maybe better to leave it
as an inspiring Ghost... Or there would be a desire (by someone alien) to control that code
now useless. (People do that. Maybe the motivation behind the Printer's firm abusive request)
The listing, for the ROM, is more than enough for us. It's what the ROM is... now tweaked.

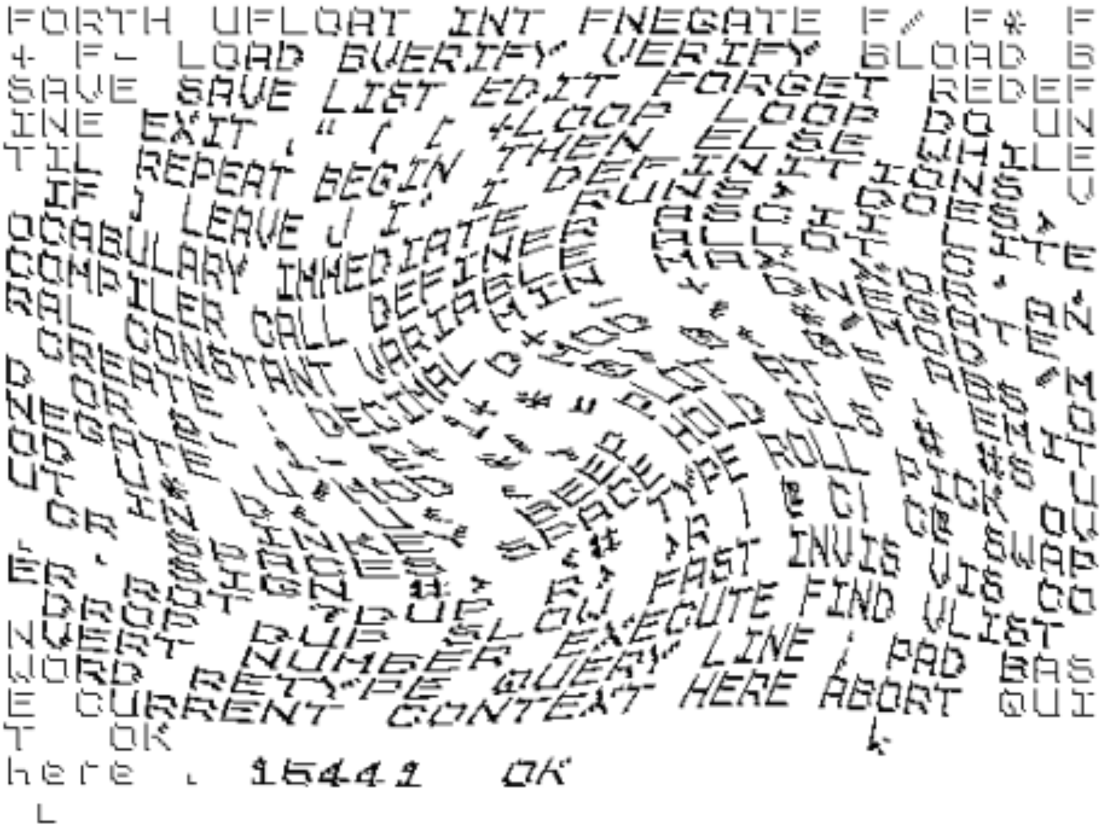
Final words ... (Ours, not Forth... To go forth!)
Private is private, more than just code. As such, our own reasoning become... We say No!
We know the confusions between object, know-how and sight.Between access and possession.
Availability turns 'right' to abuse. We have seen all that, experienced all that.
...*For too long!*

Thus:
The Jupiter is dead! But the ACE example remains: Let's honour it instead to possess hardware.
Nor to repeat (or steal) a know-how. Lames do all that to fill voids. We detect 'fetching'
results feeds abuses everywhere. It also blocks creativeness. (To Reach, not to fetch)

___
 # Cannot kill those who are already dead (those speak higher). Also:
What is known should not be written, for a soul does not fit on a description.

-

--- End of Book 1 ---



ACE-FORTH (original)

Reconstruction of the Output Listing

+ CPUs Comparisons
+ Z80 Ops Codes Chart

= Volume 2 =

USEFUL DOCUMENTS

Withhold source code only when you're ashamed of it

Chuck Moore
August 1989
Originator of Forth and owner of Computer Cowboys

Pre-conference prelude, the "Future of …" is a catchall for everything having to do with Forth. Its current place in the world is impossible to determine, and largely irrelevant. Forth is a valuable tool ---and will remain so--- regardless of the number using it. Recently I was obliged to use conventional CAD software. I am dismayed that it hasn't evolved from the 60's. Forth is the only hope for improved software, ignoring the ever-hopeful AI and neural nets, Computers are getting ever-more complicated, in violation of the first principle of human activity: "Keep it Simple."

In respect for this unique forum --- 25 words or less --- I offer the following statements to challange/guide question/comment:

1. I like classic Forth.
2. This includes BLOCKS --- simpler, faster, better than files.
3. VOCABULARY has been misused by fig-FORTH. It is a poor substitute for fast compile.
4. Forth must evolve. Standards are very dangerous.
5. ANSI committee deserves thanks for "above and beyond call of duty." Theirs is the the impossible dream.
6. Marvelous opportunity for non-ANSI Forths.
7. Forth architecture is superb for micro (macro) computers. Many variants should be explored.
8. Three keys are necessary and sufficient. QWERTY is a joke.
9. Marvelous opportunity for non-IBM PCs.
10. Work smart, not hard --- forethought.
11. A program that can do everything (ie, SPICE) can do nothing well, fast, easily.
12. PUSH and POP are better names for >R and R>.
13. Multiply is a much-over-used arithmetic operation (ie.,FFT can be replaced by Walsh-Hadamard).
14. Floating point is a bad joke.
15. Withhold source code only when you're ashamed of it.

Forth is the best computer language. I'll be using it another 20 years, with a few changes.

Point 8 is an analogy as the "25 words or less" is, over a missing context.
And maybe a joke too, Polish 'Timing' driven. ~ DuLac

```
                              ;***************************************************************
                              ;*  ACE.MAC                                                    *
                              ;*                  ROM FOR THE JUPITER ACE                    *
                              ;*                  =======================                    *
                              ;*                                                             *
                              ;*  09.12.81    S.VICKERS       FINAL 8K VERSION               *
                              ;***************************************************************

                              ;===============================================================
                              ;       CONSTANTS

                              ;KEYCHAR CODES
0001                          KLT     EQU     001H    ;LEFT ARROW
0002                          LOK     EQU     002H    ;CAPS LOCK
0003                          KRT     EQU     003H    ;RIGHT ARROW
0004                          GFX     EQU     004H    ;GRAPHIC
0005                          CDL     EQU     005H    ;DELETE CHAR
0007                          KUP     EQU     007H    ;UP ARROW
0008                          INV     EQU     008H    ;INVERT
0009                          KDN     EQU     009H    ;DOWN ARROW
000A                          LDL     EQU     00AH    ;ERASE LINE
000D                          CCR     EQU     00DH    ;END-OF-LINE
0060                          PND     EQU     060H    ;STERLING POUND
007F                          CPR     EQU     07FH    ;COPYRIGHT

0080                          CINV    EQU     080H    ;INVERTED BIT
0080                          CLAST   EQU     080H    ;LAST CHAR OF A STRING

0040                          IMM     EQU     040H    ;WORD IS "IMMEDIATE"

000C                          SAFETY  EQU     12      ;SECURITY GAP FOR THE PARAMETER-STACK

0080                          FSIGN   EQU     080H    ;MANTISSA SIGN
0040                          FEOFFS  EQU     040H    ;EXPONENT OFFSET

                              ;===============================================================
                              ;       IN AND OUT, ONLY A0 IS DECODED

00FE                          IO      EQU     0FEH    ;SPEAKER ON-FF
                                                      ;D0..4  SPLITED HORIZONTAL KEYS
                                                      ;       (VERTICAL IN A15..A8)
                                                      ;D5     CASSETTE (EAR, INPUT)
                                                      ;OUT SPEAKER OFF-FF
                                                      ;D3     CASSETTE (MIC, OUTPUT)

                              ;===============================================================
                              ;       SCREEN MEMORY (1 KBYTE)
                              ;
                              ;        VERTICAL :  24  +  4/7  +   1  +  4/7       (60/50 HZ)
                              ;       HORIZONTAL:  32  +   8   +   4  +   8
                              ;                  IMAGE + FRONT + SYNC + AFTER

2400                          SCREEN          EQU     02400H  ;24 LINES WITH 32 CHARS
2700                          SCREND          EQU     SCREEN+24*32

2701                          PADMEM          EQU     02701H  ;FREE RAM FOLLOWING
```

```
2301                    FPADMEM    EQU    PADMEM AND NOT 00400H   ;NO WAITS

2800                    SCRMEND    EQU    02800H  ;END

                        ;=================================================================
                        ;      FONT DATA (1 KBYTE)

2C00                    CHRSET  EQU    02C00H         ;128 CHARACTERS OF 8 BYTES

                        ;=================================================================
                        ;      RAM MEMORY (FOR 1 KBYTE)

3C00                    MEMBEG     EQU    03C00H  ;FIRST AVAILABLE RAM-ADDRESS

3C00                    FPWS       EQU    03C00H  ;SPACE FOR FLOATS CALCULATION

3C13                    LISTWS     EQU    03C13H  ;LIST/EDIT WORKSPACE
3C13                    LPICNT     EQU    03C13H  ;LIST/EDIT WORD COUNTER
3C14                    LPIBUF     EQU    03C14H  ;LIST/EDIT BUFFER-TABS
3C15                    LPIACT     EQU    03C15H  ;LIST/EDIT ACTUAL-TABS
3C16                    LPLCNT     EQU    03C16H  ;LIST/EDIT CHAR COUNTER

3C18                    RAMTOP     EQU    03C18H  ;FIRST NON-EXISTENT ADDRESS

3C1A                    HLD        EQU    03C1AH  ;POINTER DURING "#"

3C1C                    SCRPOS     EQU    03C1CH  ;OUTPUT-FIELD CURSOR
3C1E                    INSCRN     EQU    03C1EH  ;INPUT-FIELD BEGIN
3C20                    CURSOR     EQU    03C20H  ;INPUT-FIELD CURSOR
3C22                    ENDBUF     EQU    03C22H  ;INPUT-FIELD END

3C24                    RAMVAR     EQU    03C24H  ;INITIALIZED AFTER HERE -----

3C24                    LHALF      EQU    03C24H  ;OUTPUT-FIELD END

3C26                    KEYCOD     EQU    03C26H  ;PRESSED KEY
3C27                    KEYCNT     EQU    03C27H  ;TIMES COUNTER
3C28                    STATIN     EQU    03C28H  ;0 RELEASED INPUT
                                                  ;1 CAPS LOCK
                                                  ;2 GRAFIC
                                                  ;3 INVERSE
                                                  ;5 "ENTER" PRESSED

3C29                    EXWRCH     EQU    03C29H  ;ALTERNATIVE OUPUT

3C2B                    FRAMES     EQU    03C2BH  ;VSYNCS COUNTER

3C2F                    XCOORD     EQU    03C2FH  ;PLOT-COORDINATES
3C30                    YCOORD     EQU    03C30H  ;

3C31                    VCURRENT   EQU    03C31H  ;PTR CURRENT DICT.
3C33                    VCONTEXT   EQU    03C33H  ;PTR SEARCH DICT.
3C35                    VOCLNK     EQU    03C35H  ;PTR PREVIOUS DICT.
3C37                    STKBOT     EQU    03C37H  ;PTR FREE ABOVE DICT.
3C39                    DICT       EQU    03C39H  ;PRT LAST IN DICT.
3C3B                    SPARE      EQU    03C3BH  ;PTR ABOVE DATA-STACK
```

```
3C3D                        ERRNO           EQU     03C3DH  ;ERROR NUMBER

3C3E                        FLAGS           EQU     03C3EH  ;2 COMPILE-MODE
                                                            ;3 EDIT-MODE
                                                            ;4 INVISIBLE INPUT
                                                            ;6 COMPILER ("[","]")

3C3F                        VBASE           EQU     03C3FH  ;NUMERIC BASE

3C40                        DICT1ST         EQU     03C40H  ;DICTIONARY "FORTH"

                            ;=================================================================
                            ;       STRUCTURES:
                            ;
                            ;       DICTIONARY:
                            ;       DB...   NAME IN ASCII, LAST CHAR HAS BIT 7 = 1
                            ;       DW      LINK TO PREVIOUS DICTIONARY
                            ;       DW      LAST ADDRESS
                            ;       DB      NAME LENGHT
                            ;       DW,DW   SWITCH TO WORD
                            ;       DB      ALWAYS 0
                            ;       DW      FIRST ADDRESS
                            ;
                            ;       ROM-WORD:
                            ;       DB...   NAME IN ASCII, LAST CHAR HAS BIT 7 = 1
                            ;       DW      LINK TO PREVIOUS WORD
                            ;       DB      NAME LENGHT
                            ;       DW      FIRST CODE-ADDRESS
                            ;       ...     MORE DATA
                            ;
                            ;       RAM-WORD:
                            ;       DB...   NAME IN ASCII, LETZTES ZEICHEN HAT BIT 7 = 1
                            ;       DW      NUMBER OF BYTES TO END-OF-WORD
                            ;       DW      LINK TO PREVIOUS WORD
                            ;       DB      NAME SIZE (BIT 6 = "IMMEDIATE")
                            ;       DW      FIRST CODE-ADDRESS
                            ;       ...     MORE DATA
                            ;
                            ;       FLOATS:
                            ;       3 BYTES MANTISSE BCD
                            ;       11 BYTE EXPONENT, OFFSET 40H, BIT 7=SIGNAL
                            ;
                            ;=================================================================
                            ;       ERROR NUMBERS

FFFF                        ERRNONE EQU     -1      ;NO ERROR
0001                        ERRMEM  EQU     1       ;MEMORY FULL
0002                        ERRSTK  EQU     2       ;STACK-UNDERFOW (TO MANY DROP'S)
0003                        ERRBRK  EQU     3       ;INTERRUPTED BY THE USER
0004                        ERRIMM  EQU     4       ;IMMEDIATE-WORD IN INTERPRETER-MODE
0005                        ERRBLK  EQU     5       ;BLOCK-ERROR (EX. "IF" - "ENDIF")
0006                        ERRNAME EQU     6       ;NAME-SIZE TOO LONG IN "CRHEADER"
0007                        ERRPICK EQU     7       ;WRONG STACK-OFFSET (EX. IN "PICK")
0008                        ERRFLT  EQU     8       ;FLOAT-OVERFLOW
0009                        ERRAT   EQU     9       ;ERROR IN "AT"
```

```
000A                            ERRREAD EQU     10      ;ERROR BY "?READ" OR BY "?VERIFY"
000B                            ERRDICT EQU     11      ;DICT-ERROR, IN "REDEFINE" & "FORGET"
000C                            ERRMODE EQU     12      ;COMPILE-MODE ERROR, BY "LINKHERE"
000D                            ERRFIND EQU     13      ;WORD NOT FOUND
000E                            ERRLIST EQU     14      ;WORD NOT LISTABLE BY "LIST"

                                ;=================================================================
                                ;       RESET

                                        ORG     00000H

0000'   F3                              DI                      ;DISABLE INTERRUPTS

0001'   21 3C00                         LD      HL,MEMBEG
0004'   3E FC                           LD      A,0FCH          ;ADDRESS PAGE AND TEST VALUE
0006'   18 20                           JR      RMEMLP
                                ;=================================================================
                                ;       OUTPUT A CHAR

                                        ORG     00008H

                                RSTEMIT MACRO
                                        RST     008H
                                        ENDM

0008'   D9                              EXX
0009'   DD CB 3E 5E                     BIT     3,(IX+FLAGS-MEMBEG)
000D'   C3 03EE'                        JP      REMIT
                                ;=================================================================
                                ;       PUSH VALUE IN DE TO THE PARAMETER STACK

                                        ORG     00010H

                                RSTPUSH MACRO
                                        RST     010H
                                        ENDM

0010'                           CPUSH:
0010'   2A 3C3B                         LD      HL,(SPARE)
0013'   73                              LD      (HL),E
0014'   23                              INC     HL
0015'   C3 085F'                        JP      RPUSH
                                ;=================================================================
                                ;       POP VALUE IN PARAMETER STACK INTO DE

                                        ORG     00018H

                                RSTPULL MACRO
                                        RST     018H
                                        ENDM

0018'                           CPULL:
0018'   2A 3C3B                         LD      HL,(SPARE)
001B'   2B                              DEC     HL
001C'   56                              LD      D,(HL)
001D'   C3 0859'                        JP      RPULL
```

```
                                ;===================================================================
                                ;       REPORT AN ERROR

                                        ORG     00020H

                                RSTERR  MACRO   ERRNUM
                                        RST     020H
                                        DB      ERRNUM
                                        ENDM

0020'   E1                              POP     HL
0021'   7E                              LD      A,(HL)
0022'   32 3C3D                         LD      (ERRNO),A       ;GET ERROR NUMBER
0025'   C3 00AD'                        JP      RABORT
                                ;===================================================================
0028'                           RMEMLP:
0028'   24                              INC     H
0029'   77                              LD      (HL),A
002A'   BE                              CP      (HL)
002B'   28 FB                           JR      Z,RMEMLP        ;FIND RAM END

002D'   A4                              AND     H
002E'   67                              LD      H,A             ;ONLY FULL KBYTES

002F'   22 3C18                         LD      (RAMTOP),HL     ;SAVE END

0032'   F9                              LD      SP,HL           ;SET CPU STACK

0033'   21 010D'                        LD      HL,ROMVAR
0036'   18 03                           JR      RGOON
                                ;===================================================================
                                ;       VSYNC-INTERRUPT

                                        ORG     00038H

0038'   C3 013A'                        JP      VSYNC
                                ;===================================================================
003B'                           RGOON:
003B'   11 3C24                         LD      DE,RAMVAR
003E'   01 002D                         LD      BC,ROMVEND-ROMVAR
0041'   ED B0                           LDIR                    ;PRESET VARIABLES

0043'   DD 21 3C00                      LD      IX,MEMBEG
0047'   FD 21 04C8'                     LD      IY,RSLNEXT      ;SET POINTER

004B'   CD 0A24'                        CALL    CCLS
004E'   AF                              XOR     A
004F'   32 2700                         LD      (SCREEN+24*32),A        ;END OF PATTERNS
                                ;-------------------------------------------------------------------
0052'   21 2C00                         LD      HL,CHRSET
0055'                           RGFXLP:
0055'   7D                              LD      A,L
0056'   E6 BF                           AND     0BFH            ;4 BLOCKS GRAPHICS SET
0058'   0F                              RRCA
0059'   0F                              RRCA                    ;XX0000XX       00
005A'   0F                              RRCA                    ;XX0001XX       00
```

```
005B'   30 02                   JR      NC,RGFXM        ;XX0010XX       0F
005D'   0F                      RRCA                    ;XX0011XX       00
005E'   0F                      RRCA                    ;XX0100XX       F0
005F'                   RGFXM:                          ;XX0101XX       00
005F'   0F                      RRCA                    ;XX0110XX       FF
0060'   47                      LD      B,A             ;XX0111XX       00
0061'   9F                      SBC     A,A             ;XX1000XX       00
0062'   CB 18                   RR      B               ;XX1001XX       0F
0064'   47                      LD      B,A             ;XX1010XX       0F
0065'   9F                      SBC     A,A             ;XX1011XX       0F
0066'   A8                      XOR     B               ;XX1100XX       F0
0067'   E6 F0                   AND     0F0H            ;XX1101XX       0F
0069'   A8                      XOR     B               ;XX1110XX       FF
006A'   77                      LD      (HL),A          ;XX1111XX       0F
006B'   2C                      INC     L
006C'   20 E7                   JR      NZ,RGFXLP       ;NOT YET ALL GRAPHIC CHARS?

006E'   11 2FFF                 LD      DE,CHRSET+128*8-1
0071'   21 1FFB'                LD      HL,ROMCHR-1
0074'   01 0008                 LD      BC,8            ;8 ROWS
0077'   ED B8                   LDDR                    ;COPYRIGHT-CHAR

0079'   EB                      EX      DE,HL
007A'   3E 5F                   LD      A,128-020H-1    ;NUMBER OF REMAINING CHARS
007C'                   RCHRLP:
007C'   0E 07                   LD      C,7             ;7 ROWS
007E'   CB 6F                   BIT     5,A
0080'   28 03                   JR      Z,RCHR7         ;CHARS WITH 7 LINES?
0082'   70                      LD      (HL),B
0083'   2B                      DEC     HL
0084'   0D                      DEC     C               ;LOWER LINE BACKGROUND
0085'                   RCHR7:
0085'   EB                      EX      DE,HL
0086'   ED B8                   LDDR                    ;COPY SIGN
0088'   EB                      EX      DE,HL

0089'   70                      LD      (HL),B
008A'   2B                      DEC     HL              ;UPPER LINE BACKGROUND

008B'   3D                      DEC     A
008C'   20 EE                   JR      NZ,RCHRLP       ;NOT YET ALL CHARS?

008E'   ED 56                   IM      1               ;VSYNC ON RST 38H
0090'   18 09                   JR      RQUIT
                        ;=================================================================
0092'   51 55 49 D4             DB      'QUI','T' OR CLAST
0096'   0000                    DW      0
0098'   04                      DB      4
0099'                   QUIT:
0099'   009B'                   DW      $+2

009B'                   RQUIT:
009B'   ED 7B 3C18              LD      SP,(RAMTOP)     ;RESET CPU STACK

009F'   FB                      EI                      ;ENABLE INTERRUPTS
```

```
00A0'   C3 04F2'                      JP      QUITLOOP        ;KEEP DOING IT
                              ;=================================================================
00A3'   41 42 4F 52                   DB      'ABOR','T' OR CLAST
00A7'   D4
00A8'   0098'                         DW      QUIT-1
00AA'   05                            DB      5
00AB'                         ABORT:
00AB'   00AD'                         DW      $+2

00AD'                         RABORT:
00AD'   FD E5                         PUSH    IY
00AF'   FD 21 04B9'                   LD      IY,NEXT         ;NORMAL ERROR CHECKING

00B3'   2A 3C37                       LD      HL,(STKBOT)
00B6'   22 3C3B                       LD      (SPARE),HL      ;RESET DATA STACK

00B9'   21 3C3E                       LD      HL,FLAGS
00BC'   7E                            LD      A,(HL)
00BD'   E6 B3                         AND     NOT ((1 SHL 6) OR (1 SHL 3) OR (1 SHL 2))
00BF'   CB 56                         BIT     2,(HL)
00C1'   77                            LD      (HL),A          ;COMPILER AND EDITOR OFF
00C2'   28 1A                         JR      Z,ABGOON        ;NO COMPILER MODE?

00C4'   CD 04B9'                      CALL    NEXT
00C7'   0490' 08B3'                   DW      DP,AT,GETBYTE
00CB'   104B'
00CD'   05                            DB      5
00CE'   0DD2' 086B'                   DW      PLUS,DUP,RESCURR        ;RESET CURRENT
00D2'   1610'
00D4'   15B5' 1011'                   DW      NFA,GETWORD,STKBOT
00D8'   3C37
00DA'   08C1'                         DW      EXCLAM                  ;RESET STACK
00DC'   1A0E'                         DW      SEMICODE

00DE'                         ABGOON:
00DE'   DD CB 3D 7E                   BIT     7,(IX+ERRNO-MEMBEG)
00E2'   20 1B                         JR      NZ,ABORTEND     ;NO ERROR POSTED?

00E4'   CD 1808'                      CALL    ROMTXT
00E7'   45 52 52 4F                   DB      'ERRO','R' OR CLAST
00EB'   D2

00EC'   CD 04B9'                      CALL    NEXT
00EF'   1011' 3C3D                    DW      GETWORD,ERRNO,CAT,PNT,CR
00F3'   0896' 09B3'
00F7'   0A95'
00F9'   1A0E'                         DW      SEMICODE        ;REPORT ERRORS

00FB'   DD 36 3D FF                   LD      (IX+ERRNO-MEMBEG),ERRNONE   ;CLEAR ERRORS

00FF'                         ABORTEND:
00FF'   2A 3C37                       LD      HL,(STKBOT)
0102'   01 000C                       LD      BC,SAFETY
0105'   09                            ADD     HL,BC
0106'   22 3C3B                       LD      (SPARE),HL
0109'   FD E1                         POP     IY
```

```
010B'   18 8E                   JR      RQUIT
                        ;=============================================================
010D'                   ROMVAR:
010D'   26E0                    DW      SCREEN+23*32                    ;LHALF
010F'   00 00                   DB      0,0                             ;KEYCOD
0111'   00                      DB      0                               ;STATIN
0112'   0000                    DW      0                               ;EXWRCH
0114'   00 00 00 00             DB      0,0,0,0                         ;FRAMES
0118'   00 00                   DB      0,0                             ;XCOORD/YCOORD
011A'   3C4C                    DW      FORTH+2+RAMVAR-ROMVAR           ;VCURRENT
011C'   3C4C                    DW      FORTH+2+RAMVAR-ROMVAR           ;VCONTEXT
011E'   3C4F                    DW      FORTH+5+RAMVAR-ROMVAR           ;VOCLNK
0120'   3C51                    DW      FREEMEM                         ;STKBOT
0122'   3C45                    DW      FORTH-5+RAMVAR-ROMVAR           ;DICT
0124'   3C5D                    DW      FREEMEM+SAFETY                  ;SPARE
0126'   FF                      DB      -1                              ;ERRNO
0127'   00                      DB      0                               ;FLAGS
0128'   0A                      DB      10                              ;VBASE

0129'   46 4F 52 54             DB      'FORT','H' OR CLAST             ;DICT1ST
012D'   C8
012E'   0000 1FFF               DW      0000H,1FFFH
0132'   05                      DB      5
0133'                   FORTH:
0133'   11B5'                   DW      SETCONTEXT
0135'   3C49                    DW      FORTH-1+RAMVAR-ROMVAR   ;CONTEXT IS FORTH
0137'   00                      DB      0
0138'   0000                    DW      0

013A'                   ROMVEND:
3C51                    FREEMEM EQU     ROMVEND+RAMVAR-ROMVAR           ;AVAILABLE MEMORY
                        ;=============================================================
013A'                   VSYNC:
013A'   F5                      PUSH    AF
013B'   08                      EX      AF,AF'
013C'   F5                      PUSH    AF
013D'   C5                      PUSH    BC
013E'   D5                      PUSH    DE
013F'   E5                      PUSH    HL              ;SAVE REGISTERS

0140'   06 3E                   LD      B,62
0142'                   VDELAY:
0142'   10 FE                   DJNZ    VDELAY          ;WAIT SOME TIME

0144'   21 3C2B                 LD      HL,FRAMES
0147'                   VSCNT:
0147'   34                      INC     (HL)
0148'   23                      INC     HL
0149'   28 FC                   JR      Z,VSCNT         ;INCREASE VSYNC COUNTER

014B'   CD 0310'                CALL    VKEY            ;GET KEY WITH AUTOREPEAT

014E'   21 3C28                 LD      HL,STATIN
0151'   CB 46                   BIT     0,(HL)
0153'   28 21                   JR      Z,VSEND         ;INPUT REGISTERED?
0155'   A7                      AND     A
```

```
0156'   28 1E                   JR      Z,VSEND         ;NO KEY?
0158'   FE 20                   CP      ' '
015A'   38 14                   JR      C,VSCTRL        ;SPECIAL KEY?

015C'   CB 4E                   BIT     1,(HL)
015E'   C4 0807'                CALL    NZ,TOUPPER      ;"CAPS LOCK" ?
0161'   CB 56                   BIT     2,(HL)
0163'   28 02                   JR      Z,VSNOGRF
0165'   E6 9F                   AND     09FH            ;"GRAPHICS" ?
0167'                   VSNOGRF:
0167'   CB 5E                   BIT     3,(HL)
0169'   28 02                   JR      Z,VSNOINV
016B'   F6 80                   OR      CINV            ;"INVERSE" ?
016D'                   VSNOINV:
016D'   CD 0196'                CALL    DCDCNORM        ;DISPLAYABLE KEY

0170'                   VSCTRL:
0170'   CD 01E6'                CALL    DOCTRL          ;SPECIAL KEY

0173'   CD 0282'                CALL    DCSETCUR        ;SET CURSOR
0176'                   VSEND:
0176'   E1                      POP     HL
0177'   D1                      POP     DE
0178'   C1                      POP     BC
0179'   F1                      POP     AF
017A'   08                      EX      AF,AF'
017B'   F1                      POP     AF              ;RESTORE REGISTERS

017C'   FB                      EI                      ;ENABLE INTERRUPTS
017D'   C9                      RET
                        ;=================================================================
017E'                   DCDOCHAR:
017E'   FE 0D                   CP      CCR
0180'   20 14                   JR      NZ,DCDCNORM     ;NOT "ENTER"?

0182'   21 2700                 LD      HL,SCREEN+24*32
0185'   22 3C22                 LD      (ENDBUF),HL
0188'   22 3C20                 LD      (CURSOR),HL     ;INPUT POINTER TO SCREEN-END

018B'   AF                      XOR     A
018C'   CD 0198'                CALL    DCDCINS         ;SET NEW INPUT-END

018F'   21 26E0                 LD      HL,SCREEN+23*32
0192'   22 3C1E                 LD      (INSCRN),HL     ;ONE LINE INPUT
0195'   C9                      RET

0196'                   DCDCNORM:
0196'   A7                      AND     A
0197'   C8                      RET     Z               ;NO KEYS?

0198'                   DCDCINS:
0198'   08                      EX      AF,AF'          ;SAVE CHAR

0199'   2A 3C22                 LD      HL,(ENDBUF)
019C'   7E                      LD      A,(HL)
019D'   A7                      AND     A
```

```
019E'   28 06                   JR      Z,DCDCSCROL
01A0'   11 D900                 LD      DE,-(SCREEN+24*32)
01A3'   19                      ADD     HL,DE
01A4'   30 28                   JR      NC,DCDCEND      ;INPUT-END BEFORE SCREEN-END?

01A6'                   DCDCSCROL:
01A6'   ED 5B 3C24              LD      DE,(LHALF)
01AA'   21 DBA0                 LD      HL,-(SCREEN+3*32)
01AD'   19                      ADD     HL,DE
01AE'   30 34                   JR      NC,DCDCQUIT     ;OUTPUT-END IN FIRST 3 LINES?

01B0'   2A 3C1C                 LD      HL,(SCRPOS)
01B3'   01 0020                 LD      BC,32
01B6'   09                      ADD     HL,BC
01B7'   ED 52                   SBC     HL,DE
01B9'   D5                      PUSH    DE
01BA'   D4 0421'                CALL    NC,SCROLLUP     ;OUTPUT CURSOR IN LAST LINE?

01BD'   CD 02B0'                CALL    DCSTREND
01C0'   D1                      POP     DE
01C1'   CD 042F'                CALL    INSLINE         ;SCROLL-UP SCREEN

01C4'   21 3C1E                 LD      HL,INSCRN
01C7'   06 04                   LD      B,4             ;4-TIMES
01C9'                   DCDCSLOOP:
01C9'   CD 0443'                CALL    DECLINE
01CC'   10 FB                   DJNZ    DCDCSLOOP       ;CHANGE INPUT-START

01CE'                   DCDCEND:
01CE'   CD 0302'                CALL    DCGETCIN
01D1'   54                      LD      D,H
01D2'   5D                      LD      E,L
01D3'   23                      INC     HL
01D4'   22 3C22                 LD      (ENDBUF),HL     ;CHANGE INPUT-END

01D7'   2B                      DEC     HL
01D8'   2B                      DEC     HL
01D9'   28 02                   JR      Z,DCDCSTORE     ;INPUT CURSOR AT END?

01DB'   ED B8                   LDDR                    ;MOVE REMAINING INPUT

01DD'                   DCDCSTORE:
01DD'   08                      EX      AF,AF'
01DE'   12                      LD      (DE),A          ;RESTORE CHAR
01DF'   13                      INC     DE
01E0'   ED 53 3C20              LD      (CURSOR),DE     ;STORE NEW INPUT ADDRESS
01E4'                   DCDCQUIT:
01E4'   AF                      XOR     A               ;SET Z-FLAG, NO MORE CHARS
01E5'   C9                      RET
                        ;===============================================================
01E6'                   DOCTRL:
01E6'   21 01F0'                LD      HL,DCJMPTAB
01E9'   16 00                   LD      D,0
01EB'   5F                      LD      E,A
01EC'   19                      ADD     HL,DE           ; POINTER TO TABLE ENTRY
```

```
01ED'   5E                      LD      E,(HL)
01EE'   19                      ADD     HL,DE
01EF'   E9                      JP      (HL)            ;JUMP ADDRESS
01F0'                   DCJMPTAB:
01F0'   20                      DB      DCNOP-$         ;0 (NO KEY)
01F1'   13                      DB      DCLEFT-$        ;1 ARROW LEFT
01F2'   0C                      DB      DCFLAG-$        ;2 CAPS LOCK
01F3'   1E                      DB      DCRIGHT-$       ;3 ARROW RIGHT
01F4'   0A                      DB      DCFLAG-$        ;4 GRAPHICS
01F5'   37                      DB      DCCHARDEL-$     ;5 DELETE CHAR
01F6'   1A                      DB      DCNOP-$         ;6 (UNUSED)
01F7'   50                      DB      DCUP-$          ;7 ARROW UP
01F8'   06                      DB      DCFLAG-$        ;8 INVERTED
01F9'   9C                      DB      DCDOWN-$        ;9 ARROW DOWN
01FA'   C9                      DB      DCLINEDEL-$     ;A DELETE LINE
01FB'   15                      DB      DCNOP-$         ;B (UNUSED)
01FC'   14                      DB      DCNOP-$         ;C (UNUSED)
01FD'   D3                      DB      DCENTER-$       ;D END-OF-LINE
                        ;-----------------------------------------------------------------
01FE'                   DCFLAG:
01FE'   21 3C28                 LD      HL,STATIN
0201'   AE                      XOR     (HL)
0202'   77                      LD      (HL),A          ;CHANGE FLAG
0203'   C9                      RET
                        ;-----------------------------------------------------------------
0204'                   DCLEFT:
0204'   2A 3C20                 LD      HL,(CURSOR)
0207'   2B                      DEC     HL
0208'   7E                      LD      A,(HL)
0209'   A7                      AND     A
020A'   C8                      RET     Z               ;AT INPUT-START?

020B'   22 3C20                 LD      (CURSOR),HL     ;SAVE NEW ADDRESS

020E'   23                      INC     HL
020F'   77                      LD      (HL),A          ;MOVE CHAR
0210'                   DCNOP:
0210'   C9                      RET
                        ;-----------------------------------------------------------------
0211'                   DCRIGHT:
0211'   2A 3C20                 LD      HL,(CURSOR)
0214'   23                      INC     HL
0215'   ED 5B 3C22              LD      DE,(ENDBUF)
0219'   A7                      AND     A
021A'   ED 52                   SBC     HL,DE
021C'   C8                      RET     Z               ;AT INPUT-END?

021D'   19                      ADD     HL,DE
021E'   22 3C20                 LD      (CURSOR),HL     ;SAVE NEW ADDRESS

0221'   7E                      LD      A,(HL)
0222'   2B                      DEC     HL
0223'   77                      LD      (HL),A          ;MOVE CHAR
0224'   C9                      RET
                        ;-----------------------------------------------------------------
0225'                   DCCURDEL:
```

```
0225'   2A 3C20                  LD      HL,(CURSOR)
0228'   23                       INC     HL
0229'   22 3C20                  LD      (CURSOR),HL     ;INCREMENT INPUT ADDRESS

022C'                    DCCHARDEL:
022C'   CD 0302'                 CALL    DCGETCIN
022F'   62                       LD      H,D
0230'   6B                       LD      L,E
0231'   1B                       DEC     DE
0232'   1A                       LD      A,(DE)
0233'   A7                       AND     A
0234'   C8                       RET     Z               ;AT INPUT-START?

0235'   ED 53 3C20               LD      (CURSOR),DE
0239'   78                       LD      A,B
023A'   B1                       OR      C
023B'   28 02                    JR      Z,DCCDGOON      ;AT INPUT-END?

023D'   ED B0                    LDIR                    ;DELETE CHAR LEFT
023F'                    DCCDGOON:
023F'   2B                       DEC     HL
0240'   36 20                    LD      (HL),' '        ;CLEAR LAST CHARACTER
0242'   22 3C22                  LD      (ENDBUF),HL

0245'   0C                       INC     C               ;CLEAR Z FLAG
0246'   C9                       RET
                         ;-----------------------------------------------------------------
0247'                    DCUP:
0247'   CD 0204'                 CALL    DCLEFT
024A'   28 08                    JR      Z,DCUSCROLL     ;AT INPUT-START?

024C'   06 1F                    LD      B,31
024E'                    DCUPLOOP:
024E'   CD 0204'                 CALL    DCLEFT
0251'   10 FB                    DJNZ    DCUPLOOP        ;AT LEAST ONE LINE BACK
0253'   C9                       RET

0254'                    DCUSCROLL:
0254'   2A 3C1E                  LD      HL,(INSCRN)
0257'   ED 5B 3C24               LD      DE,(LHALF)
025B'   A7                       AND     A
025C'   ED 52                    SBC     HL,DE
025E'   C8                       RET     Z               ;INPUT START AT OUTPUT END?

025F'   CD 0225'                 CALL    DCCURDEL

0262'   2A 3C1E                  LD      HL,(INSCRN)
0265'   11 FFE0                  LD      DE,-32
0268'   AF                       XOR     A
0269'                    DCUSLOOP:
0269'   19                       ADD     HL,DE
026A'   BE                       CP      (HL)
026B'   20 FC                    JR      NZ,DCUSLOOP     ;SEARCH NEXT TAG

026D'   22 3C1E                  LD      (INSCRN),HL
```

```
0270'   CD 02F4'                        CALL    DCSETEND
0273'   22 3C20                         LD      (CURSOR),HL     ;STORE NEW INPUT-END
                                ;-----------------------------------------------------------------
0276'                           DCOUTCUR:
0276'   3E A0                           LD      A,' ' OR CINV
0278'   CD 017E'                        CALL    DCDOCHAR        ;PRINT CURSOR CHAR

027B'   2A 3C20                         LD      HL,(CURSOR)
027E'   2B                              DEC     HL
027F'   22 3C20                         LD      (CURSOR),HL     ;CORRECT ADDRESS

0282'                           DCSETCUR:
0282'   2A 3C20                         LD      HL,(CURSOR)
0285'   3A 3C28                         LD      A,(STATIN)
0288'   1F                              RRA
0289'   36 97                           LD      (HL),017H OR CINV       ;"NORMAL"
028B'   1F                              RRA
028C'   30 02                           JR      NC,SCNOCAPS
028E'   36 C3                           LD      (HL),'C' OR CINV        ;"CAPS LOCK"
0290'                           SCNOCAPS:
0290'   1F                              RRA
0291'   D0                              RET     NC
0292'   36 C7                           LD      (HL),'G' OR CINV        ;"GRAPHIC"
0294'   C9                              RET
                                ;-----------------------------------------------------------------
0295'                           DCDOWN:
0295'   CD 0211'                        CALL    DCRIGHT
0298'   28 08                           JR      Z,DCDSCROLL     ;AT INPUT-END?
029A'   06 1F                           LD      B,31
029C'                           DCDNLOOP:
029C'   CD 0211'                        CALL    DCRIGHT
029F'   10 FB                           DJNZ    DCDNLOOP        ;A LINE AT MOST
02A1'   C9                              RET

02A2'                           DCDSCROLL:
02A2'   CD 02B0'                        CALL    DCSTREND
02A5'   E0                              RET     PO              ;END FOUND?
02A6'   E5                              PUSH    HL

02A7'   CD 0225'                        CALL    DCCURDEL

02AA'   E1                              POP     HL
02AB'   CD 02ED'                        CALL    DCSETBEG        ;SET NEW INPUT-START

02AE'   18 C6                           JR      DCOUTCUR
                                ;-----------------------------------------------------------------
02B0'                           DCSTREND:
02B0'   21 2700                         LD      HL,SCREEN+24*32
02B3'   ED 5B 3C1E                      LD      DE,(INSCRN)
02B7'   A7                              AND     A
02B8'   ED 52                           SBC     HL,DE
02BA'   44                              LD      B,H
02BB'   4D                              LD      C,L             ;CALCULATE COUNT

02BC'   EB                              EX      DE,HL
02BD'   23                              INC     HL              ;POINTER BEHIND START
```

```
02BE'   AF                      XOR     A
02BF'   ED B1                   CPIR                    ;SEARCH FOR END-OF-TEXT

02C1'   2B                      DEC     HL              ;CORRECT POINTER
02C2'   C9                      RET
                        ;-----------------------------------------------------------------
02C3'                   DCLINEDEL:
02C3'   2A 3C22                 LD      HL,(ENDBUF)
02C6'   2B                      DEC     HL
02C7'   22 3C20                 LD      (CURSOR),HL     ;POINT TO INPUT-END

02CA'                   DCLDLOOP:
02CA'   CD 022C'                CALL    DCCHARDEL
02CD'   20 FB                   JR      NZ,DCLDLOOP     ;DELETE UNTIL START
02CF'   C9                      RET
                        ;-----------------------------------------------------------------
02D0'                   DCENTER:
02D0'   21 3C28                 LD      HL,STATIN
02D3'   CB EE                   SET     5,(HL)          ;SET AN "ENTER"
02D5'   CB 86                   RES     0,(HL)          ;CLEAR INPUT KEY
02D7'   C9                      RET
                        ;-----------------------------------------------------------------
02D8'                   DCCLEAR:
02D8'   21 2700                 LD      HL,SCREEN+24*32
02DB'   ED 5B 3C24              LD      DE,(LHALF)
02DF'   CD 07FA'                CALL    BLANKS          ;CLEAR INPUT FIELD

02E2'   21 26E0                 LD      HL,SCREEN+23*32
02E5'   22 3C24                 LD      (LHALF),HL
02E8'   36 00                   LD      (HL),0          ;RESET INPUT-START

02EA'                   DCRETYPE:
02EA'   2A 3C24                 LD      HL,(LHALF)

02ED'                   DCSETBEG:
02ED'   22 3C1E                 LD      (INSCRN),HL     ;SAVE NEW START
02F0'   23                      INC     HL
02F1'   22 3C20                 LD      (CURSOR),HL     ;STORE CURRENT POSITION

02F4'                   DCSETEND:
02F4'   CD 02B0'                CALL    DCSTREND
02F7'   3E 20                   LD      A,' '
02F9'                   DCSELOOP:
02F9'   2B                      DEC     HL
02FA'   BE                      CP      (HL)
02FB'   28 FC                   JR      Z,DCSELOOP      ;FIND STRING END

02FD'   23                      INC     HL
02FE'   22 3C22                 LD      (ENDBUF),HL     ;SET INPUT END
0301'   C9                      RET
                        ;-----------------------------------------------------------------
0302'                   DCGETCIN:
0302'   2A 3C22                 LD      HL,(ENDBUF)
0305'   ED 5B 3C20              LD      DE,(CURSOR)
0309'   A7                      AND     A
```

```
030A'   ED 52                   SBC     HL,DE
030C'   44                      LD      B,H
030D'   4D                      LD      C,L             ;CALCULATE COUNT

030E'   19                      ADD     HL,DE           ;RESTORE POINTER
030F'   C9                      RET
                        ;----------------------------------------------------------------
0310'                   VKEY:
0310'   CD 0336'                CALL    KEYGET
0313'   47                      LD      B,A             ;GET PRESSED KEY

0314'   2A 3C26                 LD      HL,(KEYCOD)
0317'   AD                      XOR     L
0318'   28 0B                   JR      Z,VKAGAIN       ;SAME KEY STILL PRESSED?
031A'   AD                      XOR     L
031B'   28 03                   JR      Z,VKNEW         ;NO BUTTON PRESSED?
031D'   AF                      XOR     A
031E'   BD                      CP      L
031F'   C0                      RET     NZ              ;GOT ANOTHER KEY?

0320'                   VKNEW:
0320'   68                      LD      L,B             ;SAVE THE KEY
0321'   26 20                   LD      H,32            ;LOAD TIME COUNTER
0323'   18 0D                   JR      VKQUIT

0325'                   VKAGAIN:
0325'   25                      DEC     H               ;COUNT TIME DOWN

0326'   7C                      LD      A,H
0327'   FE 1E                   CP      30
0329'   28 06                   JR      Z,VKPRESS       ;KEY RELEASED?

032B'   AF                      XOR     A
032C'   BC                      CP      H
032D'   20 03                   JR      NZ,VKQUIT       ;AUTOREPEAT TIME REACHED?

032F'   26 04                   LD      H,4             ;RESET COUNTER
0331'                   VKPRESS:
0331'   7D                      LD      A,L             ;ACCEPT KEY
0332'                   VKQUIT:
0332'   22 3C26                 LD      (KEYCOD),HL
0335'   C9                      RET
                        ;----------------------------------------------------------------
0336'                   KEYGET:
0336'   01 FEFE                 LD      BC,IO OR (0FEH SHL 8)   ;MASK AND ADDRESS
0339'   ED 50                   IN      D,(C)           ;LINE WITH "SHIFT" AND "SYMBOL"
033B'   5A                      LD      E,D             ;SAVE

033C'   CB 3A                   SRL     D
033E'   9F                      SBC     A,A
033F'   E6 D8                   AND     -40             ;OFFSET FOR NO "SHIFT",
0341'   CB 3A                   SRL     D
0343'   38 02                   JR      C,KEYGNC        ;NO "SYMBOL" ?
0345'   3E 28                   LD      A,40            ;KEY COUNT
0347'                   KEYGNC:
0347'   C6 57                   ADD     A,2*40+7        ;NORMAL "SHIFT" "SYMBOL"
```

```
0349'   6F                      LD      L,A             ;  47      87      127

034A'   7B                      LD      A,E
034B'   F6 03                   OR      3               ;LINE WITHOUT "SHIFT" AND "SYMBOL"

034D'   1E FF                   LD      E,0FFH          ;NO KEY YET
034F'                   KEYGLP:
034F'   2F                      CPL
0350'   E6 1F                   AND     1FH
0352'   57                      LD      D,A             ;MASK KEYS
0353'   28 0D                   JR      Z,KEYGNK        ;NO KEY PRESSED?

0355'   7D                      LD      A,L
0356'   1C                      INC     E
0357'   20 12                   JR      NZ,KEYGQU       ;ALREADY PRESSED?
0359'                   KEYGSC:
0359'   D6 08                   SUB     8               ;ADJUST OFFSET
035B'   CB 3A                   SRL     D
035D'   30 FA                   JR      NC,KEYGSC       ;KEY NOT YET FOUND?

035F'   5F                      LD      E,A             ;SAVE OFFSET
0360'   20 09                   JR      NZ,KEYGQU       ;MORE KEYS PRESSED?
0362'                   KEYGNK:
0362'   2D                      DEC     L               ;ADJUST OFFSET
0363'   CB 00                   RLC     B
0365'   30 06                   JR      NC,KEYGQU2      ;KEYBOARD DONE?

0367'   ED 78                   IN      A,(C)           ;GET NEXT ROW
0369'   18 E4                   JR      KEYGLP

036B'                   KEYGQU:
036B'   1E FF                   LD      E,-1            ;NO KEY PRESSED
036D'                   KEYGQU2:
036D'   7B                      LD      A,E
036E'   3C                      INC     A
036F'   C8                      RET     Z               ;NO KEY PRESSED?

0370'   21 0376'                LD      HL,KEYTBL
0373'   19                      ADD     HL,DE
0374'   7E                      LD      A,(HL)          ; GET KEY CODE
0375'   C9                      RET
0376'                   KEYTBL:
0376'   76 68 79 36             DB      'v','h','y','6','5','t','g','c' ;NORMAL
037A'   35 74 67 63
037E'   62 6A 75 37             DB      'b','j','u','7','4','r','f','x'
0382'   34 72 66 78
0386'   6E 6B 69 38             DB      'n','k','i','8','3','e','d','z'
038A'   33 65 64 7A
038E'   6D 6C 6F 39             DB      'm','l','o','9','2','w','s',0
0392'   32 77 73 00
0396'   20 0D 70 30             DB      ' ',CCR,'p','0','1','q','a',0
039A'   31 71 61 00

039E'   56 48 59 07             DB      'V','H','Y',KUP,KLT,'T','G','C' ;WITH "SHIFT"
03A2'   01 54 47 43
03A6'   42 4A 55 09             DB      'B','J','U',KDN,INV,'R','F','X'
```

```
03AA'   08 52 46 58
03AE'   4E 4B 49 03                 DB      'N','K','I',KRT,'3','E','D','Z'
03B2'   33 45 44 5A
03B6'   4D 4C 4F 04                 DB      'M','L','O',GFX,LOK,'W','S',0
03BA'   02 57 53 00
03BE'   20 0D 50 05                 DB      ' ',CCR,'P',CDL,LDL,'Q','A',0
03C2'   0A 51 41 00

03C6'   2F 5E 5B 26                 DB      '/','^','[','&','%','>','}','?' ;WITH "SYMBOL"
03CA'   25 3E 7D 3F
03CE'   2A 2D 5D 27                 DB      '*','-',']','''','$','<','{',PND
03D2'   24 3C 7B 60
03D6'   2C 2B 7F 28                 DB      ',','+',CPR,'(','#','E','\',':'
03DA'   23 45 5C 3A
03DE'   2E 3D 3B 29                 DB      '.','=',';',')','@','W','|',0
03E2'   40 57 7C 00
03E6'   20 0D 22 5F                 DB      ' ',CCR,'"','_','!','Q','~',0
03EA'   21 51 7E 00
                            ;=================================================================
03EE'                       REMIT:
03EE'   28 05                       JR      Z,RENORM        ;NOT "EDIT" ?
03F0'   CD 017E'                    CALL    DCDOCHAR
03F3'   D9                          EXX
03F4'   C9                          RET

03F5'                       RENORM:
03F5'   47                          LD      B,A
03F6'   2A 3C29                     LD      HL,(EXWRCH)
03F9'   7C                          LD      A,H
03FA'   B5                          OR      L
03FB'   78                          LD      A,B
03FC'   28 01                       JR      Z,EMITSCR
03FE'   E9                          JP      (HL)            ;USE OUTPUT VECTOR?

03FF'                       EMITSCR:
03FF'   2A 3C1C                     LD      HL,(SCRPOS)
0402'   ED 5B 3C24                  LD      DE,(LHALF)
0406'   EB                          EX      DE,HL
0407'   37                          SCF
0408'   ED 52                       SBC     HL,DE
040A'   EB                          EX      DE,HL
040B'   DC 0421'                    CALL    C,SCROLLUP      ;SCROLL A LINE IF NEEDED

040E'   FE 0D                       CP      CCR
0410'   28 04                       JR      Z,ESENTER       ;"ENTER" ?

0412'   77                          LD      (HL),A          ;STORE CHAR
0413'   23                          INC     HL              ;NEXT ADDRESS
0414'   18 06                       JR      ESQUIT

0416'                       ESENTER:
0416'   23                          INC     HL
0417'   7D                          LD      A,L
0418'   E6 1F                       AND     32-1
041A'   20 FA                       JR      NZ,ESENTER      ;POINT TO NEXT LINE
```

```
041C'                          ESQUIT:
041C'   22 3C1C                        LD      (SCRPOS),HL     ;STORE CURSOR ADDRESS
041F'   D9                             EXX
0420'   C9                             RET
                               ;-----------------------------------------------------------------
0421'                          SCROLLUP:
0421'   F5                             PUSH    AF
0422'   21 3C1C                        LD      HL,SCRPOS
0425'   CD 0443'                       CALL    DECLINE         ;ADJUST THE CURSOR ADDRESS
0428'   F1                             POP     AF

0429'   2A 3C24                        LD      HL,(LHALF)
042C'   11 2420                        LD      DE,SCREEN+32    ;SCROLL OUTPUT AREA

042F'                          INSLINE:
042F'   A7                             AND     A
0430'   ED 52                          SBC     HL,DE
0432'   44                             LD      B,H
0433'   4D                             LD      C,L             ;CHARS COUNT
0434'   21 FFE0                        LD      HL,-32
0437'   19                             ADD     HL,DE
0438'   EB                             EX      DE,HL
0439'   ED B0                          LDIR                    ;SCROLL SCREEN UP

043B'   06 20                          LD      B,32
043D'                          ILLOOP:
043D'   2B                             DEC     HL
043E'   36 20                          LD      (HL),' '
0440'   10 FB                          DJNZ    ILLOOP          ;CLEAR NEW LINE
0442'   C9                             RET
                               ;-----------------------------------------------------------------
0443'                          DECLINE:
0443'   7E                             LD      A,(HL)
0444'   D6 20                          SUB     32
0446'   77                             LD      (HL),A
0447'   23                             INC     HL
0448'   30 01                          JR      NC,DLEND
044A'   35                             DEC     (HL)
044B'                          DLEND:
044B'   23                             INC     HL
044C'   C9                             RET
                               ;=================================================================
044D'                          GETVAR:
044D'   EB                             EX      DE,HL
044E'   5E                             LD      E,(HL)
044F'   16 00                          LD      D,0             ;GET OFFSET

0451'   21 3C00                        LD      HL,MEMBEG
0454'   19                             ADD     HL,DE
0455'   EB                             EX      DE,HL
                                       RSTPUSH                 ;ADDRESS ON STACK
0456'   D7                 +           RST     010H
0457'   FD E9                          JP      (IY)
                               ;=================================================================
0459'   48 45 52 C5                    DB      'HER','E' OR CLAST
045D'   00AA'                          DW      ABORT-1
```

```
045F'   04                          DB      4
0460'                       HERE:
0460'   0462'                       DW      $+2

0462'   ED 5B 3C37                  LD      DE,(STKBOT)
                                    RSTPUSH
0466'   D7              +           RST     010H
0467'   FD E9                       JP      (IY)
                            ;=================================================================
0469'   43 4F 4E 54                 DB      'CONTEX','T' OR CLAST
046D'   45 58 D4
0470'   045F'                       DW      HERE-1
0472'   07                          DB      7
0473'                       CONTEXT:
0473'   044D'                       DW      GETVAR
0475'   33                          DB      VCONTEXT-MEMBEG
                            ;=================================================================
0476'   43 55 52 52                 DB      'CURREN','T' OR CLAST
047A'   45 4E D4
047D'   0472'                       DW      CONTEXT-1
047F'   07                          DB      7
0480'                       CURRENT:
0480'   044D'                       DW      GETVAR
0482'   31                          DB      VCURRENT-MEMBEG
                            ;=================================================================
0483'   42 41 53 C5                 DB      'BAS','E' OR CLAST
0487'   047F'                       DW      CURRENT-1
0489'   04                          DB      4
048A'                       BASE:
048A'   044D'                       DW      GETVAR
048C'   3F                          DB      VBASE-MEMBEG
                            ;=================================================================
048D'                       GETFLAGS:
048D'   044D'                       DW      GETVAR
048F'   3E                          DB      FLAGS-MEMBEG
                            ;=================================================================
0490'                       DP:
0490'   044D'                       DW      GETVAR
0492'   39                          DB      DICT-MEMBEG
                            ;=================================================================
0493'   50 41 C4                    DB      'PA','D' OR CLAST
0496'   0489'                       DW      BASE-1
0498'   03                          DB      3
0499'                       PAD:
0499'   0FF5' 2701                  DW      DOCONSTANT,PADMEM
                            ;=================================================================
049D'                       NSEMICOLON:
049D'   BB                          DB      ';' OR CLAST
049E'   0498'                       DW      PAD-1
04A0'   41                          DB      1 OR IMM
04A1'                       SEMICOLON:
04A1'   1108' 04B6'                 DW      DOCOMPILER,SEMIS
04A5'   12D8'                       DW      ASSERT
04A7'   0A                          DB      10                      ;TEST CHECK VALUE
04A8'   1A0E'                       DW      SEMICODE
```

```
04AA'   21 3C3E                         LD      HL,FLAGS
04AD'   7E                              LD      A,(HL)
04AE'   E6 BB                           AND     NOT ((1 SHL 6) OR (1 SHL 2))
04B0'   77                              LD      (HL),A          ;SWITCH OFF COMPILER
04B1'   FD E9                           JP      (IY)
                                ;==================================================================
04B3'   00                              DB      0
04B4'   FFE8                            DW      NSEMICOLON-$-1
04B6'                           SEMIS:
04B6'   04B8'                           DW      RSEMIS
04B8'                           RSEMIS:
04B8'   E1                              POP     HL              ;DISCARD CURRENT POINTER

04B9'                           NEXT:
04B9'   E1                              POP     HL              ;GET POINTER
04BA'                           NEXTSUB:
04BA'   5E                              LD      E,(HL)
04BB'   23                              INC     HL
04BC'   56                              LD      D,(HL)
04BD'   23                              INC     HL
04BE'   E5                              PUSH    HL              ;GET NEXT FORTH ADDRESS
04BF'                           NEXTDE:
04BF'   EB                              EX      DE,HL
04C0'   5E                              LD      E,(HL)
04C1'   23                              INC     HL
04C2'   56                              LD      D,(HL)
04C3'   23                              INC     HL
04C4'   EB                              EX      DE,HL
04C5'   E9                              JP      (HL)            ;JUMP TO MACHINE CODE
                                ;==================================================================
04C6'                           SLNEXT:
04C6'   04C8'                           DW      RSLNEXT
04C8'                           RSLNEXT:
04C8'   01 000B                         LD      BC,11
04CB'   ED 5B 3C3B                      LD      DE,(SPARE)
04CF'   2A 3C37                         LD      HL,(STKBOT)
04D2'   09                              ADD     HL,BC
04D3'   ED 52                           SBC     HL,DE
04D5'   38 02                           JR      C,RSLNGOON      ;STILL SPACE BETWEEN STACKS?
04D7'                           ERRORSTK:
                                        RSTERR  ERRSTK
04D7'   E7                  +           RST     020H
04D8'   02                  +           DB      ERRSTK

04D9'                           RSLNGOON:
04D9'   01 0000                         LD      BC,0
04DC'   CD 0F8C'                        CALL    MEMCHECK
04DF'   CD 04E4'                        CALL    USERBREAK
04E2'   18 D5                           JR      NEXT
                                ;==================================================================
04E4'                           USERBREAK:
04E4'   3E FE                           LD      A,0FEH
04E6'   DB FE                           IN      A,(IO)          ;READ KEYBOARD LINE
04E8'   1F                              RRA
04E9'   D8                              RET     C               ;"SHIFT" NOT PRESSED?
04EA'   3E 7F                           LD      A,7FH
```

```
04EC'   DB FE                   IN      A,(IO)          ;READ KEYBOARD LINE
04EE'   1F                      RRA
04EF'   D8                      RET     C               ;"BREAK" NOT PRESSED?
04F0'                   BREAK:
                                RSTERR  ERRBRK
04F0'   E7              +       RST     020H
04F1'   03              +       DB      ERRBRK
                        ;================================================================
04F2'                   QUITLOOP:
04F2'   CD 04B9'                CALL    NEXT
04F5'                   QLLOOP:
04F5'   058C'                   DW      QUERY                   ;GET A LINE
04F7'   0506'                   DW      LINE                    ;INTERPRET IT
04F9'   0536'                   DW      OK                      ;AND SEND "OK"
04FB'   1276' FFF7              DW      DOREPEAT,QLLOOP-$-1     ;FOREVER...
                        ;================================================================
04FF'   4C 49 4E C5             DB      'LIN','E' OR CLAST
0503'   04A0'                   DW      SEMICOLON-1
0505'   04                      DB      4
0506'                   LINE:
0506'   0EC3'                   DW      DOCOL
0508'                   LINELOOP:
0508'   04C6'                   DW      SLNEXT                  ;CHECK ALL
050A'   063D' 08EE'             DW      FIND,QDUP               ;SEARCH WORD
050E'   1283' 0007              DW      DOIF,LINENUM-$-1        ;NOT FOUND?
0512'   054F'                   DW      CHKIMM                  ;PROCESS WORD
0514'   1276' FFF1              DW      DOREPEAT,LINELOOP-$-1
0518'                   LINENUM:
0518'   06A9' 08EE'             DW      NUMBER,QDUP             ;SEARCH NUMBER
051C'   1283' 0007              DW      DOIF,LINESTR-$-1        ;NOT FOUND?
0520'   0564'                   DW      CHKNUMBER               ;PROCESS NUMBER
0522'   1276' FFE3              DW      DOREPEAT,LINELOOP-$-1
0526'                   LINESTR:
0526'   061B' 0C1A'             DW      CHKSTRING,ZEROEQ        ;SEARCH TEXT
052A'   1283' 0003              DW      DOIF,LINEERR-$-1        ;NOT FOUND?
052E'   04B6'                   DW      SEMIS
0530'                   LINEERR:
0530'   0578'                   DW      RETYPE                  ;REPORT ERRORS
0532'   1276' FFD3              DW      DOREPEAT,LINELOOP-$-1
                        ;================================================================
0536'                   OK:
0536'   0538'                   DW      $+2

0538'   3A 3C3E                 LD      A,(FLAGS)
053B'   CB 77                   BIT     6,A
053D'   20 0E                   JR      NZ,OKQUIT       ;COMPILER STILL RUNNING?
053F'   CB 67                   BIT     4,A
0541'   20 0A                   JR      NZ,OKQUIT       ;INPUT INVISIBLE?

0543'   CD 1808'                CALL    ROMTXT
0546'   20 4F 4B A0             DB      ' OK',' ' OR CLAST
054A'   3E 0D                   LD      A,CCR
                                RSTEMIT
054C'   CF              +       RST     008H

054D'                   OKQUIT:
```

```
054D'   FD E9                       JP      (IY)
                            ;================================================================
054F'                       CHKIMM:
054F'   0551'                       DW      $+2

                                    RSTPULL                 ;CODE FIELD ADDRESS
0551'   DF              +           RST     018H
0552'   1B                          DEC     DE
0553'   1A                          LD      A,(DE)
0554'   2F                          CPL
0555'   DD A6 3E                    AND     (IX+FLAGS-MEMBEG)
0558'   E6 40                       AND     1 SHL 6
055A'   13                          INC     DE
055B'   28 04                       JR      Z,CHKIQUIT      ;COMPILER OFF OR IMMEDIATE?

                                    RSTPUSH
055D'   D7              +           RST     010H
055E'   11 0F4E'                    LD      DE,KOMMA
0561'                       CHKIQUIT:
0561'   C3 04BF'                    JP      NEXTDE
                            ;----------------------------------------------------------------
0564'                       CHKNUMBER:
0564'   0566'                       DW      $+2

                                    RSTPULL
0566'   DF              +           RST     018H
0567'   DD CB 3E 76                 BIT     6,(IX+FLAGS-MEMBEG)
056B'   20 F4                       JR      NZ,CHKIQUIT     ;COMPILER ON?
056D'   FD E9                       JP      (IY)
                            ;================================================================
056F'   52 45 54 59                 DB      'RETYP','E' OR CLAST
0573'   50 C5
0575'   058B'                       DW      QUERY-1
0577'   06                          DB      6
0578'                       RETYPE:
0578'   057A'                       DW      $+2

057A'   CD 02EA'                    CALL    DCRETYPE
057D'   CD 0276'                    CALL    DCOUTCUR
0580'   36 BF                       LD      (HL),'?' OR CINV        ;CHANGE CURSOR
0582'   18 10                       JR      QSTART
                            ;================================================================
0584'   51 55 45 52                 DB      'QUER','Y' OR CLAST
0588'   D9
0589'   0505'                       DW      LINE-1
058B'   05                          DB      5
058C'                       QUERY:
058C'   058E'                       DW      $+2

058E'   CD 02D8'                    CALL    DCCLEAR
0591'   CD 0276'                    CALL    DCOUTCUR

0594'                       QSTART:
0594'   21 3C28                     LD      HL,STATIN
0597'   CB C6                       SET     0,(HL)          ;ENABLE INPUT
0599'   CB AE                       RES     5,(HL)          ;NO "ENTER" YET
```

```
059B'                           QLOOP:
059B'   CB 6E                           BIT     5,(HL)
059D'   28 FC                           JR      Z,QLOOP         ;WAIT FOR "ENTER"

059F'   CD 0225'                        CALL    DCCURDEL
05A2'   FD E9                           JP      (IY)
                                ;===============================================================
05A4'   57 4F 52 C4                     DB      'WOR','D' OR CLAST
05A8'   0577'                           DW      RETYPE-1
05AA'   04                              DB      4
05AB'                           WORD:
05AB'   05AD'                           DW      $+2

                                        RSTPULL                 ;GET DELIMITER
05AD'   DF              +               RST     018H

05AE'   21 27FE                         LD      HL,SCRMEND-2
05B1'   06 FD                           LD      B,SCRMEND-SCREND-3
05B3'                           WCLLOOP:
05B3'   36 20                           LD      (HL),' '
05B5'   2B                              DEC     HL
05B6'   10 FB                           DJNZ    WCLLOOP         ;CLEAR BUFFER

05B8'   D5                              PUSH    DE
05B9'   EB                              EX      DE,HL
                                        RSTPUSH
05BA'   D7              +               RST     010H
05BB'   D1                              POP     DE
05BC'   CD 05E1'                        CALL    CWORD           ;READ TEXT

05BF'   04                              INC     B
05C0'   05                              DEC     B
05C1'   28 03                           JR      Z,WGOON1
05C3'   01 00FF                         LD      BC,255          ;LIMIT COUNT TO 255
05C6'                           WGOON1:
05C6'   21 2701                         LD      HL,PADMEM
05C9'   71                              LD      (HL),C          ;STORE COUNT
05CA'   23                              INC     HL
05CB'   3E FC                           LD      A,252
05CD'   B9                              CP      C
05CE'   30 01                           JR      NC,WGOON2
05D0'   4F                              LD      C,A             ;LIMIT COUNT
05D1'                           WGOON2:
05D1'   0C                              INC     C
05D2'   D5                              PUSH    DE
05D3'   C5                              PUSH    BC
05D4'   EB                              EX      DE,HL
05D5'   ED B0                           LDIR                    ;MOVE INPUT
05D7'   C1                              POP     BC
05D8'   D1                              POP     DE
05D9'   0D                              DEC     C
05DA'   CD 07DA'                        CALL    BLWORD          ;CLEAR ENTRY
05DD'   FD E9                           JP      (IY)
                                ;===============================================================
05DF'                           GETSTRING:
05DF'   1E 20                           LD      E,' '           ;SPACE CHAR IS DELIMITER
```

```
05E1'                       CWORD:
05E1'   2A 3C24                     LD      HL,(LHALF)
05E4'   22 3C1E                     LD      (INSCRN),HL

05E7'   01 0000                     LD      BC,0            ;NO CHAR YET
05EA'                       CWLOOP1:
05EA'   23                          INC     HL
05EB'   7E                          LD      A,(HL)
05EC'   BB                          CP      E
05ED'   28 FB                       JR      Z,CWLOOP1       ;SEARCH START
05EF'   A7                          AND     A
05F0'   28 0E                       JR      Z,CWNFND

05F2'   E5                          PUSH    HL              ;SAVE START
05F3'                       CWLOOP2:
05F3'   03                          INC     BC              ;COUNT
05F4'   23                          INC     HL
05F5'   7E                          LD      A,(HL)
05F6'   A7                          AND     A
05F7'   28 03                       JR      Z,CWEND         ;END OF TEXT?
05F9'   BB                          CP      E
05FA'   20 F7                       JR      NZ,CWLOOP2      ;SEARCH END
05FC'                       CWEND:
05FC'   D1                          POP     DE              ;RESTORE START

05FD'   AF                          XOR     A
05FE'   B8                          CP      B
05FF'   C9                          RET                     ;TEST FOR COUNT=256

0600'                       CWNFND:
0600'   D5                          PUSH    DE
0601'   CD 02B0'                    CALL    DCSTREND
0604'   E2 0614'                    JP      PO,CWERR        ;ENTRY END FOUND?
0607'   ED 5B 3C24                  LD      DE,(LHALF)
060B'   CD 07FA'                    CALL    BLANKS          ;CLEAR ENTRY AREA
060E'   22 3C24                     LD      (LHALF),HL
0611'   D1                          POP     DE
0612'   18 CD                       JR      CWORD           ;NEXT WORD

0614'                       CWERR:
0614'   EB                          EX      DE,HL           ;POINTER TO END
0615'   C1                          POP     BC
0616'   01 0000                     LD      BC,0
0619'   37                          SCF                     ;REPORT FAILURE
061A'   C9                          RET
                            ;=================================================================
061B'                       CHKSTRING:
061B'   061D'                       DW      $+2

061D'   CD 05DF'                    CALL    GETSTRING
0620'   50                          LD      D,B
0621'   59                          LD      E,C
                                    RSTPUSH
0622'   D7              +           RST     010H
0623'   FD E9                       JP      (IY)
                            ;=================================================================
```

```
0625'   56 4C 49 53                     DB      'VLIS','T' OR CLAST
0629'   D4
062A'   05AA'                           DW      WORD-1
062C'   05                              DB      5
062D'                           VLIST:
062D'   062F'                           DW      $+2

062F'   3E 0D                           LD      A,CCR
                                        RSTEMIT
0631'   CF              +               RST     008H

0632'   0E 00                           LD      C,0             ;FIND ALL WORDS
0634'   18 0E                           JR      RFIND
                                ;================================================================
0636'   46 49 4E C4                     DB      'FIN','D' OR CLAST
063A'   062C'                           DW      VLIST-1
063C'   04                              DB      4
063D'                           FIND:
063D'   063F'                           DW      $+2

063F'   CD 05DF'                        CALL    GETSTRING
0642'   38 46                           JR      C,RZERO         ;NO WORD ENTERED?

0644'                           RFIND:
0644'   2A 3C33                         LD      HL,(VCONTEXT)
0647'   7E                              LD      A,(HL)
0648'   23                              INC     HL
0649'   66                              LD      H,(HL)
064A'   6F                              LD      L,A             ;GET THE FIRST POINTER
064B'                           FLOOP:
064B'   7E                              LD      A,(HL)
064C'   E6 3F                           AND     3FH
064E'   28 2F                           JR      Z,FNEXT2        ;NO MORE WORDS?
0650'   A9                              XOR     C
0651'   28 04                           JR      Z,FTEST         ;SAME LENGTH?
0653'   79                              LD      A,C
0654'   A7                              AND     A
0655'   20 28                           JR      NZ,FNEXT2       ;SEARCHING SINGLE WORD?
0657'                           FTEST:
0657'   D5                              PUSH    DE
0658'   E5                              PUSH    HL
0659'   CD 15E8'                        CALL    PTR2NAME
065C'   B1                              OR      C
065D'   28 17                           JR      Z,FPRINT        ;PRINT WORD IMMEDIATELY?

065F'   41                              LD      B,C             ;GET WORD LENGTH
0660'                           FCOMPARE:
0660'   1A                              LD      A,(DE)
0661'   CD 0807'                        CALL    TOUPPER
0664'   13                              INC     DE
0665'   AE                              XOR     (HL)
0666'   E6 7F                           AND     NOT CLAST
0668'   23                              INC     HL
0669'   20 12                           JR      NZ,FNEXT1       ;DIFFERENT WORD?
066B'   10 F3                           DJNZ    FCOMPARE        ;NOT YET ALL CHARS?
```

```
066D'   D1                              POP     DE
066E'   13                              INC     DE
                                        RSTPUSH                 ;POINTER TO CODE FIELD
066F'   D7              +               RST     010H
0670'   D1                              POP     DE
0671'   CD 07DA'                        CALL    BLWORD          ;CLEAR ENTRY IF REQUIRED
0674'   FD E9                           JP      (IY)

0676'                           FPRINT:
0676'   CD 17FB'                        CALL    OUTTXT
0679'   76                              HALT                    ;WAIT FOR VSYNC
067A'   CD 04E4'                        CALL    USERBREAK
067D'                           FNEXT1:
067D'   E1                              POP     HL
067E'   D1                              POP     DE
067F'                           FNEXT2:
067F'   2B                              DEC     HL
0680'   7E                              LD      A,(HL)
0681'   2B                              DEC     HL
0682'   6E                              LD      L,(HL)
0683'   67                              LD      H,A             ;NEXT POINTER
0684'   B5                              OR      L
0685'   20 C4                           JR      NZ,FLOOP        ;NOT YET ALL WORDS?
0687'   C3                              DB      0C3H            ;JP RZERO (HRM-HRM !!!)
                                ;=================================================================
0688'                           ZERO:
0688'   068A'                           DW      $+2

068A'                           RZERO:
068A'   11 0000                         LD      DE,0
                                        RSTPUSH
068D'   D7              +               RST     010H
068E'   FD E9                           JP      (IY)
                                ;=================================================================
0690'   45 58 45 43                     DB      'EXECUT','E' OR CLAST
0694'   55 54 C5
0697'   063C'                           DW      FIND-1
0699'   07                              DB      7
069A'                           EXECUTE:
069A'   069C'                           DW      $+2

                                        RSTPULL
069C'   DF              +               RST     018H
069D'   C3 04BF'                        JP      NEXTDE
                                ;=================================================================
06A0'   4E 55 4D 42                     DB      'NUMBE','R' OR CLAST
06A4'   45 D2
06A6'   0699'                           DW      EXECUTE-1
06A8'   06                              DB      6
06A9'                           NUMBER:
06A9'   06AB'                           DW      $+2

06AB'   CD 05DF'                        CALL    GETSTRING
06AE'   38 DA                           JR      C,RZERO         ;NO WORD ENTERED?

06B0'   C5                              PUSH    BC
```

```
06B1'   D5                      PUSH    DE
06B2'   CD 074C'                CALL    CNVINT
06B5'   20 05                   JR      NZ,NFLOAT       ;NO SPACE?
06B7'   11 1006'                LD      DE,LITERAL
06BA'   18 58                   JR      NUMBERQUIT      ;16-BIT-INTEGER

06BC'                   NFLOAT:
                                RSTPULL
06BC'   DF              +       RST     018H
06BD'   11 0000                 LD      DE,0
                                RSTPUSH
06C0'   D7              +       RST     010H
06C1'   11 4500                 LD      DE,0 OR ((FEOFFS+5) SHL 8)

06C4'   C1                      POP     BC
06C5'   C5                      PUSH    BC
06C6'   0A                      LD      A,(BC)
06C7'   FE 2D                   CP      '-'
06C9'   20 03                   JR      NZ,NFGOON       ;POSITIVE NUMBER?
06CB'   16 C5                   LD      D,FSIGN OR (FEOFFS+5)
06CD'   03                      INC     BC
06CE'                   NFGOON:
                                RSTPUSH
06CE'   D7              +       RST     010H
06CF'   50                      LD      D,B
06D0'   59                      LD      E,C

06D1'   2B                      DEC     HL
06D2'   2B                      DEC     HL
06D3'                   NFLOOP1:
06D3'   CD 0723'                CALL    DECGET
06D6'   23                      INC     HL
06D7'   34                      INC     (HL)
06D8'   2B                      DEC     HL
06D9'   30 F8                   JR      NC,NFLOOP1      ;CONVERT INTEGER PART

06DB'   FE FE                   CP      '.'-'0'
06DD'   20 3D                   JR      NZ,NUMBERERR    ;NOT DECIMAL POINT?
06DF'                   NFLOOP2:
06DF'   CD 0723'                CALL    DECGET
06E2'   30 FB                   JR      NC,NFLOOP2      ;CONVERT FRACTIONAL PART

06E4'   C6 30                   ADD     A,'0'
06E6'   CD 077B'                CALL    CNVEND
06E9'   20 04                   JR      NZ,NFEXP        ;NO SPACE?
06EB'   1E 00                   LD      E,0
06ED'   18 0E                   JR      NFEGOON
06EF'                   NFEXP:
06EF'   E6 DF                   AND     NOT 020H
06F1'   FE 45                   CP      'E'
06F3'   20 27                   JR      NZ,NUMBERERR    ;NO EXPONENT?
06F5'   E5                      PUSH    HL
06F6'   CD 074C'                CALL    CNVINT
                                RSTPULL
06F9'   DF              +       RST     018H
06FA'   E1                      POP     HL
```

```
06FB'   20 1F                   JR      NZ,NUMBERERR    ;NO SPACE?
06FD'                   NFEGOON:
06FD'   CD 0740'                CALL    FZEROEQ
0700'   28 0F                   JR      Z,NFQUIT        ;VALUE = 0?

0702'   23                      INC     HL
0703'   7E                      LD      A,(HL)
0704'   E6 7F                   AND     7FH
0706'   83                      ADD     A,E
0707'   FA 071C'                JP      M,NUMBERERR
070A'   28 10                   JR      Z,NUMBERERR     ;EXPONENT OUT OF RANGE
070C'   AE                      XOR     (HL)
070D'   E6 7F                   AND     7FH
070F'   AE                      XOR     (HL)            ;KEEP SIGN
0710'   77                      LD      (HL),A          ;STORE EXPONENT
0711'                   NFQUIT:
0711'   11 1055'                LD      DE,LITFLOAT
0714'                   NUMBERQUIT:
                                RSTPUSH
0714'   D7              +       RST     010H
0715'   D1                      POP     DE
0716'   C1                      POP     BC
0717'   CD 07DA'                CALL    BLWORD
071A'   FD E9                   JP      (IY)

071C'                   NUMBERERR:
071C'   E1                      POP     HL
071D'   E1                      POP     HL
                                RSTPULL
071E'   DF              +       RST     018H
                                RSTPULL
071F'   DF              +       RST     018H
0720'   C3 068A'                JP      RZERO
                        ;-----------------------------------------------------------------
0723'                   DECGET:
0723'   1A                      LD      A,(DE)
0724'   13                      INC     DE
0725'   D6 30                   SUB     '0'
0727'   D8                      RET     C
0728'   FE 0A                   CP      10
072A'   3F                      CCF
072B'   D8                      RET     C               ;CHARACTER <'0' OR> '9'?

072C'                   DECSHIN:
072C'   4F                      LD      C,A
072D'   7E                      LD      A,(HL)
072E'   E6 F0                   AND     0F0H
0730'   C0                      RET     NZ              ;TOP DIGIT <> 0?
0731'   79                      LD      A,C
0732'                   DECSTORE:
0732'   2B                      DEC     HL
0733'   2B                      DEC     HL
0734'   0E 03                   LD      C,3
0736'                   DSLOOP:
0736'   ED 6F                   RLD
0738'   23                      INC     HL
```

```
0739'   0D                      DEC     C
073A'   20 FA                   JR      NZ,DSLOOP       ;LOWER DIGIT
073C'   35                      DEC     (HL)
073D'   2B                      DEC     HL
073E'   BF                      CP      A
073F'   C9                      RET                     ;DIGIT INSERTED, TEST FOR 0
                        ;-----------------------------------------------------------------
0740'                   FZEROEQ:
0740'   06 06                   LD      B,6
0742'                   FZEQLP:
0742'   AF                      XOR     A
0743'   CD 072C'                CALL    DECSHIN
0746'   C0                      RET     NZ              ;DIGIT <> 0 FOUND?
0747'   10 F9                   DJNZ    FZEQLP          ;LIMIT CHECK TO ALL
0749'   23                      INC     HL
074A'   70                      LD      (HL),B          ;CLEAR EXPONENT
074B'   C9                      RET
                        ;-----------------------------------------------------------------
074C'                   CNVINT:
                                RSTPUSH
074C'   D7              +       RST     010H
074D'   CD 04B9'                CALL    NEXT
0750'   086B' 0896'             DW      DUP,CAT,GETBYTE
0754'   104B'
0756'   2D                      DB      '-'
0757'   0C4A'                   DW      EQ                      ;NEGATIVE SIGN?
0759'   086B' 0DA9'             DW      DUP,NEGATE,GTR
075D'   08D2'
075F'   0DD2' 0E1F'             DW      PLUS,ONEMINUS           ;ADJUST POINTER
0763'   0688' 0688'             DW      ZERO,ZERO,ROT
0767'   08FF'
0769'   078A'                   DW      CONVERT                 ;CONVERT NUMBER
076B'   08FF' 08DF'             DW      ROT,RGT,IFN0NEG         ;NEGATE IF NEEDED
076F'   0D94'
0771'   08FF' 0879'             DW      ROT,DROP                ;REMOVE HIGH WORD
0775'   0885'                   DW      SWAP
0777'   1A0E'                   DW      SEMICODE
                                RSTPULL
0779'   DF              +       RST     018H
077A'   1A                      LD      A,(DE)
077B'                   CNVEND:
077B'   FE 20                   CP      ' '
077D'   C8                      RET     Z
077E'   A7                      AND     A
077F'   C9                      RET                     ;TEST FOR SPACE
                        ;=================================================================
0780'   43 4F 4E 56             DB      'CONVER','T' OR CLAST
0784'   45 52 D4
0787'   06A8'                   DW      NUMBER-1
0789'   07                      DB      7
078A'                   CONVERT:
078A'   0EC3'                   DW      DOCOL
078C'                   CNVTLOOP:
078C'   0E09' 086B'             DW      ONEPLUS,DUP,GTR         ;KEEP ADDRESS
0790'   08D2'
0792'   0896' 07B8'             DW      CAT,CNVDIGIT            ;CONVERT A CHAR
```

```
0796'   1283' 001B                  DW      DOIF,CNVTEND-$-1        ;NO NUMBERS?
079A'   0885'                       DW      SWAP
079C'   048A' 0896'                 DW      BASE,CAT,UMUL
07A0'   0CA8'
07A2'   0879' 08FF'                 DW      DROP,ROT
07A6'   048A' 0896'                 DW      BASE,CAT,UMUL
07AA'   0CA8'
07AC'   0DEE'                       DW      DPLUS                   ;INSERT NUMBER
07AE'   08DF'                       DW      RGT                     ;GET ADDRESS
07B0'   1276' FFD9                  DW      DOREPEAT,CNVTLOOP-$-1
07B4'                       CNVTEND:
07B4'   08DF'                       DW      RGT                     ;ADJUST STACK
07B6'   04B6'                       DW      SEMIS
                            ;-----------------------------------------------------------------
07B8'                       CNVDIGIT:
07B8'   07BA'                       DW      $+2
                                    RSTPULL
07BA'   DF              +           RST     018H
07BB'   7B                          LD      A,E
07BC'   CD 0807'                    CALL    TOUPPER         ;GET CHAR
07BF'   C6 D0                       ADD     A,-'0'
07C1'   30 14                       JR      NC,CNVDQUIT     ;CHAR <'0' ?
07C3'   FE 0A                       CP      10
07C5'   38 06                       JR      C,CNVDOK        ;CHAR < '9' ?
07C7'   C6 EF                       ADD     A,'0'-'A'
07C9'   30 0C                       JR      NC,CNVDQUIT     ;CHAR < 'A' ?
07CB'   C6 0A                       ADD     A,10            ;ADJUST VALUE
07CD'                       CNVDOK:
07CD'   DD BE 3F                    CP      (IX+VBASE-MEMBEG)
07D0'   30 05                       JR      NC,CNVDQUIT     ;CHAR TOO HIGH?
07D2'   16 00                       LD      D,0
07D4'   5F                          LD      E,A
                                    RSTPUSH                 ;SAVE DIGIT
07D5'   D7              +           RST     010H
07D6'   37                          SCF
07D7'                       CNVDQUIT:
07D7'   C3 0C21'                    JP      CMPPUSH         ;SUCCESS=1 ERROR=0
                            ;=================================================================
07DA'                       BLWORD:
07DA'   62                          LD      H,D
07DB'   6B                          LD      L,E             ;POINT TO START
07DC'   03                          INC     BC
07DD'   09                          ADD     HL,BC
07DE'   E5                          PUSH    HL              ;POINTER BEHIND SEPARATOR

07DF'   DD CB 3E 66                 BIT     4,(IX+FLAGS-MEMBEG)
07E3'   CC 097F'                    CALL    Z,CTYPE         ;INPUT VISIBLE?

07E6'   CD 02B0'                    CALL    DCSTREND        ;FIND ENDING

07E9'   D1                          POP     DE
07EA'   A7                          AND     A
07EB'   ED 52                       SBC     HL,DE
07ED'   44                          LD      B,H
07EE'   4D                          LD      C,L             ; CALCULATE REMAINING CHARS
```

```
07EF'   2A 3C1E                 LD      HL,(INSCRN)
07F2'   23                      INC     HL
07F3'   EB                      EX      DE,HL
07F4'   38 05                   JR      C,BLANKS2
07F6'   28 02                   JR      Z,BLANKS
07F8'   ED B0                   LDIR                    ;ERASE INPUT
                        ;-----------------------------------------------------------------
07FA'                   BLANKS:
07FA'   A7                      AND     A
07FB'                   BLANKS2:
07FB'   ED 52                   SBC     HL,DE
07FD'   EB                      EX      DE,HL           ;CALCULATE COUNT
07FE'                   BLLOOP:
07FE'   7A                      LD      A,D
07FF'   B3                      OR      E
0800'   C8                      RET     Z               ;ALL ERASED?

0801'   36 20                   LD      (HL),' '
0803'   23                      INC     HL              ;ERASE NEXT CHARACTER
0804'   1B                      DEC     DE
0805'   18 F7                   JR      BLLOOP
                        ;=================================================================
0807'                   TOUPPER:
0807'   E6 7F                   AND     7FH
0809'   FE 61                   CP      'a'
080B'   D8                      RET     C
080C'   FE 7B                   CP      'z'+1
080E'   D0                      RET     NC
080F'   E6 5F                   AND     5FH
0811'   C9                      RET
                        ;=================================================================
0812'   56 49 D3                DB      'VI','S' OR CLAST
0815'   0789'                   DW      CONVERT-1
0817'   03                      DB      3
0818'                   VIS:
0818'   081A'                   DW      $+2

081A'   DD CB 3E A6             RES     4,(IX+FLAGS-MEMBEG)     ;INPUT VISIBLE
081E'   FD E9                   JP      (IY)
                        ;=================================================================
0820'   49 4E 56 49             DB      'INVI','S' OR CLAST
0824'   D3
0825'   0817'                   DW      VIS-1
0827'   05                      DB      5
0828'                   INVIS:
0828'   082A'                   DW      $+2

082A'   DD CB 3E E6             SET     4,(IX+FLAGS-MEMBEG)     ;INPUT INVISIBLE
082E'   FD E9                   JP      (IY)
                        ;=================================================================
0830'   46 41 53 D4             DB      'FAS','T' OR CLAST
0834'   0827'                   DW      INVIS-1
0836'   04                      DB      4
0837'                   FAST:
0837'   0839'                   DW      $+2
```

```
0839'   FD 21 04B9'                     LD      IY,NEXT
083D'   FD E9                           JP      (IY)
                                ;==================================================================
083F'   53 4C 4F D7                     DB      'SLO','W' OR CLAST
0843'   0836'                           DW      FAST-1
0845'   04                              DB      4
0846'                           SLOW:
0846'   0848'                           DW      $+2

0848'   FD 21 04C8'                     LD      IY,RSLNEXT
084C'   FD E9                           JP      (IY)
                                ;==================================================================
084E'                           PULLBC:
084E'   2A 3C3B                         LD      HL,(SPARE)
0851'   2B                              DEC     HL
0852'   46                              LD      B,(HL)
0853'   2B                              DEC     HL
0854'   4E                              LD      C,(HL)
0855'   22 3C3B                         LD      (SPARE),HL
0858'   C9                              RET
                                ;==================================================================
0859'                           RPULL:
0859'   2B                              DEC     HL
085A'   5E                              LD      E,(HL)
085B'   22 3C3B                         LD      (SPARE),HL
085E'   C9                              RET
                                ;==================================================================
085F'                           RPUSH:
085F'   72                              LD      (HL),D
0860'   23                              INC     HL
0861'   22 3C3B                         LD      (SPARE),HL
0864'   C9                              RET
                                ;==================================================================
0865'   44 55 D0                        DB      'DU','P' OR CLAST
0868'   0845'                           DW      SLOW-1
086A'   03                              DB      3
086B'                           DUP:
086B'   086D'                           DW      $+2

                                        RSTPULL
086D'   DF                  +           RST     018H
                                        RSTPUSH
086E'   D7                  +           RST     010H
                                        RSTPUSH
086F'   D7                  +           RST     010H
0870'   FD E9                           JP      (IY)
                                ;==================================================================
0872'   44 52 4F D0                     DB      'DRO','P' OR CLAST
0876'   086A'                           DW      DUP-1
0878'   04                              DB      4
0879'                           DROP:
0879'   087B'                           DW      $+2

                                        RSTPULL
087B'   DF                  +           RST     018H
087C'   FD E9                           JP      (IY)
```

```
                                ;===================================================================
087E'   53 57 41 D0                     DB      'SWA','P' OR CLAST
0882'   0878'                           DW      DROP-1
0884'   04                              DB      4
0885'                           SWAP:
0885'   0887'                           DW      $+2

                                        RSTPULL
0887'   DF              +               RST     018H
0888'   CD 084E'                        CALL    PULLBC
                                        RSTPUSH
088B'   D7              +               RST     010H
088C'   50                              LD      D,B
088D'   59                              LD      E,C
                                        RSTPUSH
088E'   D7              +               RST     010H
088F'   FD E9                           JP      (IY)
                                ;===================================================================
0891'   43 C0                           DB      'C','@' OR CLAST
0893'   0884'                           DW      SWAP-1
0895'   02                              DB      2
0896'                           CAT:
0896'   0898'                           DW      $+2

                                        RSTPULL
0898'   DF              +               RST     018H
0899'   1A                              LD      A,(DE)
089A'   5F                              LD      E,A
089B'   16 00                           LD      D,0
                                        RSTPUSH
089D'   D7              +               RST     010H
089E'   FD E9                           JP      (IY)
                                ;===================================================================
08A0'   43 A1                           DB      'C','!' OR CLAST
08A2'   0895'                           DW      CAT-1
08A4'   02                              DB      2
08A5'                           CEXCLAM:
08A5'   08A7'                           DW      $+2

                                        RSTPULL
08A7'   DF              +               RST     018H
08A8'   CD 084E'                        CALL    PULLBC
08AB'   79                              LD      A,C
08AC'   12                              LD      (DE),A
08AD'   FD E9                           JP      (IY)
                                ;===================================================================
08AF'   C0                              DB      '@' OR CLAST
08B0'   08A4'                           DW      CEXCLAM-1
08B2'   01                              DB      1
08B3'                           AT:
08B3'   08B5'                           DW      $+2

                                        RSTPULL
08B5'   DF              +               RST     018H
08B6'   EB                              EX      DE,HL
08B7'   5E                              LD      E,(HL)
```

```
08B8'   23                          INC     HL
08B9'   56                          LD      D,(HL)
                                    RSTPUSH
08BA'   D7              +           RST     010H
08BB'   FD E9                       JP      (IY)
                            ;=================================================================
08BD'   A1                          DB      '!' OR CLAST
08BE'   08B2'                       DW      AT-1
08C0'   01                          DB      1
08C1'                       EXCLAM:
08C1'   08C3'                       DW      $+2

                                    RSTPULL
08C3'   DF              +           RST     018H
08C4'   CD 084E'                    CALL    PULLBC
08C7'   EB                          EX      DE,HL
08C8'   71                          LD      (HL),C
08C9'   23                          INC     HL
08CA'   70                          LD      (HL),B
08CB'   FD E9                       JP      (IY)
                            ;=================================================================
08CD'   3E D2                       DB      '>','R' OR CLAST
08CF'   08C0'                       DW      EXCLAM-1
08D1'   02                          DB      2
08D2'                       GTR:
08D2'   08D4'                       DW      $+2

                                    RSTPULL
08D4'   DF              +           RST     018H
08D5'   C1                          POP     BC
08D6'   D5                          PUSH    DE
08D7'   C5                          PUSH    BC
08D8'   FD E9                       JP      (IY)
                            ;=================================================================
08DA'   52 BE                       DB      'R','>' OR CLAST
08DC'   08D1'                       DW      GTR-1
08DE'   02                          DB      2
08DF'                       RGT:
08DF'   08E1'                       DW      $+2

08E1'   C1                          POP     BC
08E2'   D1                          POP     DE
08E3'   C5                          PUSH    BC
                                    RSTPUSH
08E4'   D7              +           RST     010H
08E5'   FD E9                       JP      (IY)
                            ;=================================================================
08E7'   3F 44 55 D0                 DB      '?DU','P' OR CLAST
08EB'   08DE'                       DW      RGT-1
08ED'   04                          DB      4
08EE'                       QDUP:
08EE'   08F0'                       DW      $+2

                                    RSTPULL
08F0'   DF              +           RST     018H
                                    RSTPUSH
```

```
08F1'   D7              +               RST     010H
08F2'   7A                              LD      A,D
08F3'   B3                              OR      E
08F4'   C4 0010'                        CALL    NZ,CPUSH
08F7'   FD E9                           JP      (IY)
                                ;===============================================================
08F9'   52 4F D4                        DB      'RO','T' OR CLAST
08FC'   08ED'                           DW      QDUP-1
08FE'   03                              DB      3
08FF'                           ROT:
08FF'   0EC3'                           DW      DOCOL
0901'   08D2' 0885'                     DW      GTR,SWAP,RGT,SWAP
0905'   08DF' 0885'
0909'   04B6'                           DW      SEMIS
                                ;===============================================================
090B'   4F 56 45 D2                     DB      'OVE','R' OR CLAST
090F'   08FE'                           DW      ROT-1
0911'   04                              DB      4
0912'                           OVER:
0912'   0EC3'                           DW      DOCOL
0914'   08D2' 086B'                     DW      GTR,DUP,RGT,SWAP
0918'   08DF' 0885'
091C'   04B6'                           DW      SEMIS
                                ;===============================================================
091E'   50 49 43 CB                     DB      'PIC','K' OR CLAST
0922'   0911'                           DW      OVER-1
0924'   04                              DB      4
0925'                           PICK:
0925'   0927'                           DW      $+2

0927'   CD 094D'                        CALL    CPICK
092A'   FD E9                           JP      (IY)
                                ;===============================================================
092C'   52 4F 4C CC                     DB      'ROL','L' OR CLAST
0930'   0924'                           DW      PICK-1
0932'   04                              DB      4
0933'                           ROLL:
0933'   0935'                           DW      $+2

0935'   CD 094D'                        CALL    CPICK

0938'   EB                              EX      DE,HL
0939'   2A 3C37                         LD      HL,(STKBOT)
093C'   ED 52                           SBC     HL,DE
093E'   D2 04D7'                        JP      NC,ERRORSTK     ;STACK NOT THAT LARGE?

0941'   62                              LD      H,D
0942'   6B                              LD      L,E
0943'   23                              INC     HL
0944'   23                              INC     HL
0945'   ED B0                           LDIR                    ;MOVE STACK
0947'   ED 53 3C3B                      LD      (SPARE),DE
094B'   FD E9                           JP      (IY)
                                ;===============================================================
094D'                           CPICK:
094D'   CD 084E'                        CALL    PULLBC
```

```
0950'   0B                      DEC     BC
0951'   CB 21                   SLA     C
0953'   CB 10                   RL      B
0955'   03                      INC     BC
0956'   03                      INC     BC
0957'   30 02                   JR      NC,CPKGOON      ;OFFSET OK ?
                                RSTERR  ERRPICK
0959'   E7              +       RST     020H
095A'   07              +       DB      ERRPICK

095B'                   CPKGOON:
095B'   2A 3C3B                 LD      HL,(SPARE)
095E'   ED 42                   SBC     HL,BC
0960'   E5                      PUSH    HL
0961'   5E                      LD      E,(HL)
0962'   23                      INC     HL
0963'   56                      LD      D,(HL)
                                RSTPUSH                 ;NUMBER OBTAINED
0964'   D7              +       RST     010H
0965'   E1                      POP     HL
0966'   C9                      RET
                        ;=================================================================
0967'   54 59 50 C5             DB      'TYP','E' OR CLAST
096B'   0932'                   DW      ROLL-1
096D'   04                      DB      4
096E'                   TYPE:
096E'   0970'                   DW      $+2

0970'   CD 084E'                CALL    PULLBC
                                RSTPULL
0973'   DF              +       RST     018H
0974'   CD 097F'                CALL    CTYPE
0977'   FD E9                   JP      (IY)
                        ;=================================================================
0979'                   TYPEDE:
0979'   1A                      LD      A,(DE)
097A'   4F                      LD      C,A
097B'   13                      INC     DE
097C'   1A                      LD      A,(DE)
097D'   47                      LD      B,A
097E'   13                      INC     DE
                        ;-----------------------------------------------------------------
097F'                   CTYPE:
097F'   78                      LD      A,B
0980'   B1                      OR      C
0981'   C8                      RET     Z
0982'   1A                      LD      A,(DE)
0983'   13                      INC     DE
0984'   0B                      DEC     BC
                                RSTEMIT
0985'   CF              +       RST     008H
0986'   18 F7                   JR      CTYPE
                        ;=================================================================
0988'   3C A3                   DB      '<','#' OR CLAST
098A'   096D'                   DW      TYPE-1
098C'   02                      DB      2
```

```
098D'                           LTNUM:
098D'   098F'                           DW      $+2

098F'   21 27FF                         LD      HL,SCRMEND-1
0992'   22 3C1A                         LD      (HLD),HL        ;PREPARE POINTERS
0995'   FD E9                           JP      (IY)
                                ;===============================================================
0997'   23 BE                           DB      '#','>' OR CLAST
0999'   098C'                           DW      LTNUM-1
099B'   02                              DB      2
099C'                           NUMGT:
099C'   099E'                           DW      $+2

                                        RSTPULL
099E'   DF               +              RST     018H
                                        RSTPULL                 ;CLEAN STACK
099F'   DF               +              RST     018H

09A0'   ED 5B 3C1A                      LD      DE,(HLD)
                                        RSTPUSH                 ;GET POINTER
09A4'   D7               +              RST     010H

09A5'   21 27FF                         LD      HL,SCRMEND-1
09A8'   A7                              AND     A
09A9'   ED 52                           SBC     HL,DE
09AB'   EB                              EX      DE,HL
                                        RSTPUSH                 ;CALCULATE LENGTH
09AC'   D7               +              RST     010H
09AD'   FD E9                           JP      (IY)
                                ;===============================================================
09AF'   AE                              DB      '.' OR CLAST
09B0'   0A49'                           DW      SIGN-1
09B2'   01                              DB      1
09B3'                           PNT:
09B3'   0EC3'                           DW      DOCOL
09B5'   098D' 086B'                     DW      LTNUM,DUP               ;START CONVERSION
09B9'   0C0D' 0688'                     DW      ABS,ZERO                ;AS A DOUBLE WORD
09BD'   09E1'                           DW      NUMS                    ;CONVERT ABSOLUTE VALUE
09BF'   08FF' 0A4A'                     DW      ROT,SIGN                ;THEN ITS SIGN
09C3'                           PNTLEFT:
09C3'   099C'                           DW      NUMGT                   ;END CONVERSION
09C5'   096E' 0A73'                     DW      TYPE,SPACE              ;TYPE IT
09C9'   04B6'                           DW      SEMIS
                                ;===============================================================
09CB'   55 AE                           DB      'U','.' OR CLAST
09CD'   09B2'                           DW      PNT-1
09CF'   02                              DB      2
09D0'                           UPNT:
09D0'   0EC3'                           DW      DOCOL
09D2'   0688' 098D'                     DW      ZERO,LTNUM,NUMS         ;START CONVERSION
09D6'   09E1'
09D8'   1276' FFE8                      DW      DOREPEAT,PNTLEFT-$-1
                                ;===============================================================
09DC'   23 D3                           DB      '#','S' OR CLAST
09DE'   09CF'                           DW      UPNT-1
09E0'   02                              DB      2
```

```
09E1'                           NUMS:
09E1'   0EC3'                           DW      DOCOL
09E3'                           NUMSLP:
09E3'   09F7'                           DW      NUM                     ;OBTAIN ONE CHAR
09E5'   0912' 0912'                     DW      OVER,OVER,LOR,ZEROEQ
09E9'   0E36' 0C1A'
09ED'   128D' FFF3                      DW      DOUNTIL,NUMSLP-$-1      ;REMAINING <> 0?
09F1'   04B6'                           DW      SEMIS
                                ;===============================================================
09F3'   A3                              DB      '#' OR CLAST
09F4'   09E0'                           DW      NUMS-1
09F6'   01                              DB      1
09F7'                           NUM:
09F7'   0EC3'                           DW      DOCOL
09F9'   048A' 0896'                     DW      BASE,CAT,DIV32BY16,ROT  ;WITH MODULO "BASE"
09FD'   0CC4' 08FF'
0A01'   0A07' 0A5C'                     DW      NIBASC,HOLD             ;SAVED AS CHAR
0A05'   04B6'                           DW      SEMIS
                                ;===============================================================
0A07'                           NIBASC:
0A07'   0A09'                           DW      $+2

                                        RSTPULL
0A09'   DF              +               RST     018H
0A0A'   7B                              LD      A,E             ;GET NIBBLE
0A0B'   C6 30                           ADD     A,'0'
0A0D'   FE 3A                           CP      '0'+10
0A0F'   38 02                           JR      C,NADEC         ;ADJUSTEMENT FOR 'A' ...
0A11'   C6 07                           ADD     A,7
0A13'                           NADEC:
0A13'   5F                              LD      E,A
                                        RSTPUSH                 ;ASCII VALUE
0A14'   D7              +               RST     010H
0A15'   FD E9                           JP      (IY)
                                ;===============================================================
0A17'   43 4C D3                        DB      'CL','S' OR CLAST
0A1A'   09F6'                           DW      NUM-1
0A1C'   03                              DB      3
0A1D'                           CLS:
0A1D'   0A1F'                           DW      $+2

0A1F'   CD 0A24'                        CALL    CCLS
0A22'   FD E9                           JP      (IY)

0A24'                           CCLS:
0A24'   11 26FF                         LD      DE,SCREEN+24*32-1
0A27'   2A 3C24                         LD      HL,(LHALF)
0A2A'   01 0020                         LD      BC,32
0A2D'   09                              ADD     HL,BC
0A2E'   2B                              DEC     HL
0A2F'   ED B8                           LDDR                    ;LAST OUTPUT LINE TO SCREEN-END

0A31'   ED 43 3C2F                      LD      (XCOORD),BC     ;CLEAR COORDINATES

0A35'   21 2400                         LD      HL,SCREEN
0A38'   22 3C1C                         LD      (SCRPOS),HL     ;CURSOR HOME
```

```
0A3B'   13                      INC     DE
0A3C'   EB                      EX      DE,HL
0A3D'   22 3C24                 LD      (LHALF),HL      ;SET OUTPUT END

0A40'   C3 07FA'                JP      BLANKS          ;CLEAR OUTPUT AREA
                        ;=================================================================
0A43'   53 49 47 CE             DB      'SIG','N' OR CLAST
0A47'   099B'                   DW      NUMGT-1
0A49'   04                      DB      4
0A4A'                   SIGN:
0A4A'   0A4C'                   DW      $+2

                                RSTPULL
0A4C'   DF              +       RST     018H
0A4D'   CB 12                   RL      D
0A4F'   1E 2D                   LD      E,'-'
0A51'   38 0C                   JR      C,RHOLD         ;PLACE '-' IF NEEDED
0A53'   FD E9                   JP      (IY)
                        ;=================================================================
0A55'   48 4F 4C C4             DB      'HOL','D' OR CLAST
0A59'   0A1C'                   DW      CLS-1
0A5B'   04                      DB      4
0A5C'                   HOLD:
0A5C'   0A5E'                   DW      $+2

                                RSTPULL
0A5E'   DF              +       RST     018H
0A5F'                   RHOLD:
0A5F'   2A 3C1A                 LD      HL,(HLD)
0A62'   2D                      DEC     L
0A63'   28 04                   JR      Z,HOLDQUIT      ;BUFFER FULL ?

0A65'   22 3C1A                 LD      (HLD),HL
0A68'   73                      LD      (HL),E          ;SAVE CHAR
0A69'                   HOLDQUIT:
0A69'   FD E9                   JP      (IY)
                        ;=================================================================
0A6B'   53 50 41 43             DB      'SPAC','E' OR CLAST
0A6F'   C5
0A70'   0A5B'                   DW      HOLD-1
0A72'   05                      DB      5
0A73'                   SPACE:
0A73'   0A75'                   DW      $+2

0A75'   3E 20                   LD      A,' '
                                RSTEMIT
0A77'   CF              +       RST     008H
0A78'                   SPACEQUIT:
0A78'   FD E9                   JP      (IY)
                        ;=================================================================
0A7A'   53 50 41 43             DB      'SPACE','S' OR CLAST
0A7E'   45 D3
0A80'   0A72'                   DW      SPACE-1
0A82'   06                      DB      6
0A83'                   SPACES:
```

```
0A83'   0A85'                           DW      $+2

                                        RSTPULL
0A85'   DF              +               RST     018H
0A86'                           SPCLOOP:
0A86'   1B                              DEC     DE
0A87'   CB 7A                           BIT     7,D
0A89'   20 ED                           JR      NZ,SPACEQUIT    ;ALL TYPED ?
0A8B'   3E 20                           LD      A,' '
                                        RSTEMIT
0A8D'   CF              +               RST     008H
0A8E'   18 F6                           JR      SPCLOOP
                                ;=================================================================
0A90'   43 D2                           DB      'C','R' OR CLAST
0A92'   0A82'                           DW      SPACES-1
0A94'   02                              DB      2
0A95'                           CR:
0A95'   0A97'                           DW      $+2

0A97'   3E 0D                           LD      A,CCR
                                        RSTEMIT
0A99'   CF              +               RST     008H
0A9A'   FD E9                           JP      (IY)
                                ;=================================================================
0A9C'   45 4D 49 D4                     DB      'EMI','T' OR CLAST
0AA0'   0A94'                           DW      CR-1
0AA2'   04                              DB      4
0AA3'                           EMIT:
0AA3'   0AA5'                           DW      $+2

                                        RSTPULL
0AA5'   DF              +               RST     018H
0AA6'   7B                              LD      A,E
                                        RSTEMIT
0AA7'   CF              +               RST     008H
0AA8'   FD E9                           JP      (IY)
                                ;=================================================================
0AAA'   46 AE                           DB      'F','.' OR CLAST
0AAC'   0AA2'                           DW      EMIT-1
0AAE'   02                              DB      2
0AAF'                           FPNT:
0AAF'   0AB1'                           DW      $+2

0AB1'   2A 3C3B                         LD      HL,(SPARE)
0AB4'   2B                              DEC     HL
0AB5'   CB 7E                           BIT     7,(HL)
0AB7'   CB BE                           RES     7,(HL)
0AB9'   28 03                           JR      Z,FPGOON1
0ABB'   3E 2D                           LD      A,'-'
                                        RSTEMIT                 ;PRINT NEGATIVE SIGN
0ABD'   CF              +               RST     008H
0ABE'                           FPGOON1:

0ABE'   1E 00                           LD      E,0             ;NOT A EXPONENT
0AC0'   7E                              LD      A,(HL)
0AC1'   3D                              DEC     A
```

```
0AC2'   FE 49                   CP      FEOFFS+9
0AC4'   30 04                   JR      NC,FPGOON2
0AC6'   FE 3C                   CP      FEOFFS-4
0AC8'   30 04                   JR      NC,FPGOON3      ;NO EXPONENT REQUIRED?
0ACA'                   FPGOON2:
0ACA'   36 41                   LD      (HL),FEOFFS+1
0ACC'   3C                      INC     A
0ACD'   5F                      LD      E,A             ;SAVE EXPONENT
0ACE'                   FPGOON3:

0ACE'   3E 40                   LD      A,FEOFFS
0AD0'   96                      SUB     (HL)
0AD1'   38 09                   JR      C,FPMLOOP       ;NEGATIVE EXPONENT?
0AD3'   47                      LD      B,A
0AD4'   04                      INC     B
0AD5'   3E 2E                   LD      A,'.'
0AD7'                   FPH0:
                                RSTEMIT
0AD7'   CF              +       RST     008H
0AD8'   3E 30                   LD      A,'0'
0ADA'   10 FB                   DJNZ    FPH0            ;PRINT LEADING ZEROS

0ADC'                   FPMLOOP:
0ADC'   3E 40                   LD      A,'@'
0ADE'   BE                      CP      (HL)
0ADF'   9F                      SBC     A,A
0AE0'   2B                      DEC     HL
0AE1'   B6                      OR      (HL)
0AE2'   2B                      DEC     HL
0AE3'   B6                      OR      (HL)
0AE4'   2B                      DEC     HL
0AE5'   B6                      OR      (HL)
0AE6'   23                      INC     HL
0AE7'   23                      INC     HL
0AE8'   28 12                   JR      Z,FP0           ;NUMBER = 0?

0AEA'   AF                      XOR     A
0AEB'   CD 0732'                CALL    DECSTORE
0AEE'   C6 30                   ADD     A,'0'
                                RSTEMIT                 ;PRINT NEXT NUMERIC CHARS
0AF0'   CF              +       RST     008H
0AF1'   23                      INC     HL
0AF2'   7E                      LD      A,(HL)
0AF3'   FE 40                   CP      FEOFFS
0AF5'   20 E5                   JR      NZ,FPMLOOP      ;VALUE < 0.1 OR VALUE >= 1.0 ?
0AF7'   3E 2E                   LD      A,'.'
                                RSTEMIT
0AF9'   CF              +       RST     008H
0AFA'   18 E0                   JR      FPMLOOP         ;PRINT DECIMAL POINT

0AFC'                   FP0:
0AFC'   7B                      LD      A,E
0AFD'   A7                      AND     A
0AFE'   20 05                   JR      NZ,FPEXP        ;PRINTABLE EXPONENT ?
0B00'   3E 20                   LD      A,' '
                                RSTEMIT
```

```
0B02'   CF              +               RST     008H
0B03'   18 0B                           JR      FPQUIT

0B05'                           FPEXP:
0B05'   D6 41                           SUB     FEOFFS+1
0B07'   6F                              LD      L,A
0B08'   9F                              SBC     A,A
0B09'   67                              LD      H,A
0B0A'   3E 45                           LD      A,'E'
                                        RSTEMIT
0B0C'   CF              +               RST     008H
0B0D'   CD 180E'                        CALL    PNTHL           ;PRINT EXPONENT
0B10'                           FPQUIT:
                                        RSTPULL
0B10'   DF              +               RST     018H
                                        RSTPULL
0B11'   DF              +               RST     018H
0B12'   FD E9                           JP      (IY)
                                ;=================================================================
0B14'   41 D4                           DB      'A','T' OR CLAST
0B16'   0AAE'                           DW      FPNT-1
0B18'   02                              DB      2
0B19'                           ATPOS:
0B19'   0B1B'                           DW      $+2

                                        RSTPULL                 ;COLUMN
0B1B'   DF              +               RST     018H
0B1C'   CD 084E'                        CALL    PULLBC          ;LINE
0B1F'   79                              LD      A,C
0B20'   CD 0B28'                        CALL    CATPOS
0B23'   22 3C1C                         LD      (SCRPOS),HL
0B26'   FD E9                           JP      (IY)

0B28'                           CATPOS:
0B28'   C6 20                           ADD     A,32
0B2A'   6F                              LD      L,A
0B2B'   26 01                           LD      H,1             ;SCREEN / 32
0B2D'   29                              ADD     HL,HL
0B2E'   29                              ADD     HL,HL
0B2F'   29                              ADD     HL,HL
0B30'   29                              ADD     HL,HL
0B31'   29                              ADD     HL,HL           ;SCREEN + LINE
0B32'   16 00                           LD      D,0
0B34'   7B                              LD      A,E
0B35'   E6 1F                           AND     1FH
0B37'   5F                              LD      E,A
0B38'   19                              ADD     HL,DE           ;SCREEN + LINE + COLUMN

0B39'   ED 5B 3C24                      LD      DE,(LHALF)
0B3D'   ED 52                           SBC     HL,DE
0B3F'   19                              ADD     HL,DE
0B40'   D8                              RET     C               ;NOT BEHIND OUTPUT AREA?
                                        RSTERR  ERRAT
0B41'   E7              +               RST     020H
0B42'   09              +               DB      ERRAT
                                ;=================================================================
```

```
0B43'   50 4C 4F D4                DB      'PLO','T' OR CLAST
0B47'   0B18'                      DW      ATPOS-1
0B49'   04                         DB      4
0B4A'                      PLOT:
0B4A'   0B4C'                      DW      $+2

0B4C'   CD 084E'                   CALL    PULLBC          ;0/1/2/3 = RES/SET/NOP/XOR

                                   RSTPULL                 ;Y-COORDINATE
0B4F'   DF             +           RST     018H
0B50'   DD 73 30                   LD      (IX+YCOORD-MEMBEG),E
0B53'   CB 3B                      SRL     E
0B55'   CB 11                      RL      C               ;GET LSB-Y

0B57'   3E 16                      LD      A,22
0B59'   93                         SUB     E               ;Y-COORDINATE AS LINE NUMBER

                                   RSTPULL                 ;X-COORDINATE
0B5A'   DF             +           RST     018H
0B5B'   DD 73 2F                   LD      (IX+XCOORD-MEMBEG),E
0B5E'   CB 3B                      SRL     E
0B60'   CB 11                      RL      C               ;GET LSB-X

0B62'   CD 0B28'                   CALL    CATPOS          ;POSITION IN SCREEN

0B65'   7E                         LD      A,(HL)          ;GET OLD CHAR
0B66'   E6 78                      AND     78H
0B68'   FE 10                      CP      10H
0B6A'   7E                         LD      A,(HL)
0B6B'   28 02                      JR      Z,PLGOON        ;ALREADY A MOSAIC?
0B6D'   3E 10                      LD      A,10H           ;AN EMPTY MOSAIC
0B6F'                      PLGOON:
0B6F'   5F                         LD      E,A             ;KEEP MOSAIC
0B70'   16 87                      LD      D,87H           ;SET MASK

0B72'   79                         LD      A,C
0B73'   E6 03                      AND     3
0B75'   47                         LD      B,A
0B76'   28 07                      JR      Z,PLX0Y0        ;X=0 AND Y=0 ?

0B78'   2F                         CPL
0B79'   C6 02                      ADD     A,2
0B7B'   CE 03                      ADC     A,3
0B7D'   57                         LD      D,A
0B7E'   43                         LD      B,E             ;BIT MASKS FOR X <> 0 AND Y <> 0
0B7F'                      PLX0Y0:
0B7F'   79                         LD      A,C
0B80'   0F                         RRCA
0B81'   0F                         RRCA
0B82'   0F                         RRCA
0B83'   9F                         SBC     A,A             ;CLEAR / SET MASK
0B84'   CB 59                      BIT     3,C
0B86'   20 04                      JR      NZ,PLXOR        ;NOP/XOR ?
0B88'   AB                         XOR     E
0B89'   07                         RLCA
0B8A'   9F                         SBC     A,A
```

```
0B8B'   A8                          XOR     B               ;PREPARE CLEAR/SET
0B8C'                       PLXOR:
0B8C'   A2                          AND     D
0B8D'   AB                          XOR     E
0B8E'   77                          LD      (HL),A          ;SAVE NEW MOSAIC
0B8F'   FD E9                       JP      (IY)
                            ;==================================================================
0B91'   42 45 45 D0                 DB      'BEE','P' OR CLAST
0B95'   0B49'                       DW      PLOT-1
0B97'   04                          DB      4
0B98'                       BEEP:
0B98'   0EC3'                       DW      DOCOL
0B9A'   0912' 104B'                 DW      OVER,GETBYTE
0B9E'   7D                          DB      125
0B9F'   0885' 0D7A'                 DW      SWAP,MULDIV             ; ADJUST VALUE
0BA3'   1A0E'                       DW      SEMICODE

                                    RSTPULL
0BA5'   DF              +           RST     018H
0BA6'   CD 084E'                    CALL    PULLBC
0BA9'   21 00F9                     LD      HL,250-1
0BAC'   09                          ADD     HL,BC
0BAD'   2C                          INC     L               ;(??? RUNDUNG)
0BAE'   F3                          DI
0BAF'                       BLOOP:
0BAF'   3E 7F                       LD      A,7FH
0BB1'   DB FE                       IN      A,(IO)
0BB3'   0F                          RRCA
0BB4'   30 11                       JR      NC,BDBREAK      ;INTERRUPTED?
0BB6'   CD 0BC9'                    CALL    BEEPDELAY
0BB9'   1B                          DEC     DE
0BBA'   7A                          LD      A,D
0BBB'   D3 FE                       OUT     (IO),A
0BBD'   CD 0BC9'                    CALL    BEEPDELAY
0BC0'   B3                          OR      E
0BC1'   C2 0BAF'                    JP      NZ,BLOOP        ;NOT YET EXPIRED?
0BC4'   FB                          EI
0BC5'   FD E9                       JP      (IY)

0BC7'                       BDBREAK:
                                    RSTERR  ERRBRK
0BC7'   E7              +           RST     020H
0BC8'   03              +           DB      ERRBRK

0BC9'                       BEEPDELAY:
0BC9'   45                          LD      B,L
0BCA'   4C                          LD      C,H
0BCB'                       BDLOOP:
0BCB'   10 FE                       DJNZ    BDLOOP
0BCD'   05                          DEC     B
0BCE'   0D                          DEC     C
0BCF'   C2 0BCB'                    JP      NZ,BDLOOP       ;A SMALL WAIT...
0BD2'   C9                          RET
                            ;==================================================================
0BD3'   49 4E 4B 45                 DB      'INKE','Y' OR CLAST
0BD7'   D9
```

```
0BD8'   0B97'                   DW      BEEP-1
0BDA'   05                      DB      5
0BDB'                   INKEY:
0BDB'   0BDD'                   DW      $+2

0BDD'   CD 0336'                CALL    KEYGET
0BE0'   5F                      LD      E,A
0BE1'   16 00                   LD      D,0
                                RSTPUSH
0BE3'   D7              +       RST     010H
0BE4'   FD E9                   JP      (IY)
                        ;=================================================================
0BE6'   49 CE                   DB      'I','N' OR CLAST
0BE8'   0BDA'                   DW      INKEY-1
0BEA'   02                      DB      2
0BEB'                   IN:
0BEB'   0BED'                   DW      $+2

0BED'   CD 084E'                CALL    PULLBC
0BF0'   16 00                   LD      D,0
0BF2'   ED 58                   IN      E,(C)
                                RSTPUSH
0BF4'   D7              +       RST     010H
0BF5'   FD E9                   JP      (IY)
                        ;=================================================================
0BF7'   4F 55 D4                DB      'OU','T' OR CLAST
0BFA'   0BEA'                   DW      IN-1
0BFC'   03                      DB      3
0BFD'                   OUT:
0BFD'   0BFF'                   DW      $+2

0BFF'   CD 084E'                CALL    PULLBC
                                RSTPULL
0C02'   DF              +       RST     018H
0C03'   ED 59                   OUT     (C),E
0C05'   FD E9                   JP      (IY)
                        ;=================================================================
0C07'   41 42 D3                DB      'AB','S' OR CLAST
0C0A'   0BFC'                   DW      OUT-1
0C0C'   03                      DB      3
0C0D'                   ABS:
0C0D'   0EC3'                   DW      DOCOL
0C0F'   086B' 0D94'             DW      DUP,IFN0NEG
0C13'   04B6'                   DW      SEMIS
                        ;=================================================================
0C15'   30 BD                   DB      '0','=' OR CLAST
0C17'   0C0C'                   DW      ABS-1
0C19'   02                      DB      2
0C1A'                   ZEROEQ:
0C1A'   0C1C'                   DW      $+2

                                RSTPULL
0C1C'   DF              +       RST     018H
0C1D'   7A                      LD      A,D
0C1E'   B3                      OR      E
0C1F'   FE 01                   CP      1               ;C, A=0
```

```
0C21'                           CMPPUSH:
0C21'   3E 00                           LD      A,0
0C23'   57                              LD      D,A
0C24'   17                              RLA
0C25'   5F                              LD      E,A
                                        RSTPUSH                 ;C, RESULT=1, ELSE IS 0
0C26'   D7              +               RST     010H
0C27'   FD E9                           JP      (IY)
                                ;=================================================================
0C29'   30 BC                           DB      '0','<' OR CLAST
0C2B'   0C19'                           DW      ZEROEQ-1
0C2D'   02                              DB      2
0C2E'                           ZEROLT:
0C2E'   0C30'                           DW      $+2

                                        RSTPULL
0C30'   DF              +               RST     018H
0C31'   CB 12                           RL      D               ;GET SIGN
0C33'   18 EC                           JR      CMPPUSH
                                ;=================================================================
0C35'   30 BE                           DB      '0','>' OR CLAST
0C37'   0C2D'                           DW      ZEROLT-1
0C39'   02                              DB      2
0C3A'                           ZEROGT:
0C3A'   0C3C'                           DW      $+2

                                        RSTPULL
0C3C'   DF              +               RST     018H
0C3D'   7A                              LD      A,D
0C3E'   B3                              OR      E
0C3F'   28 E0                           JR      Z,CMPPUSH       ;= 0 ?
0C41'   CB 12                           RL      D
0C43'   3F                              CCF
0C44'   18 DB                           JR      CMPPUSH         ;GET INVERTED SIGN
                                ;=================================================================
0C46'   BD                              DB      '=' OR CLAST
0C47'   0C39'                           DW      ZEROGT-1
0C49'   01                              DB      1
0C4A'                           EQ:
0C4A'   0EC3'                           DW      DOCOL
0C4C'   0DE1' 0C1A'                     DW      MINUS,ZEROEQ
0C50'   04B6'                           DW      SEMIS
                                ;=================================================================
0C52'   BE                              DB      '>' OR CLAST
0C53'   0C49'                           DW      EQ-1
0C55'   01                              DB      1
0C56'                           GT:
0C56'   0C58'                           DW      $+2

                                        RSTPULL
0C58'   DF              +               RST     018H
0C59'   D5                              PUSH    DE
                                        RSTPULL
0C5A'   DF              +               RST     018H
0C5B'   E1                              POP     HL
0C5C'   CD 0C99'                        CALL    GREATER
```

```
0C5F'   18 C0                           JR      CMPPUSH
                                ;=================================================================
0C61'   BC                              DB      '<' OR CLAST
0C62'   0C55'                           DW      GT-1
0C64'   01                              DB      1
0C65'                           LT:
0C65'   0EC3'                           DW      DOCOL
0C67'   0885' 0C56'                     DW      SWAP,GT
0C6B'   04B6'                           DW      SEMIS
                                ;=================================================================
0C6D'   55 BC                           DB      'U','<' OR CLAST
0C6F'   0C64'                           DW      LT-1
0C71'   02                              DB      2
0C72'                           ULT:
0C72'   0C74'                           DW      $+2

0C74'   CD 084E'                        CALL    PULLBC
0C77'                           UCMP:
                                        RSTPULL
0C77'   DF              +               RST     018H
0C78'   EB                              EX      DE,HL
0C79'   A7                              AND     A
0C7A'   ED 42                           SBC     HL,BC           ;C = (BC > HL)
0C7C'   18 A3                           JR      CMPPUSH
                                ;=================================================================
0C7E'   44 BC                           DB      'D','<' OR CLAST
0C80'   0C71'                           DW      ULT-1
0C82'   02                              DB      2
0C83'                           DLT:
0C83'   0C85'                           DW      $+2

                                        RSTPULL
0C85'   DF              +               RST     018H
0C86'   D5                              PUSH    DE
0C87'   CD 084E'                        CALL    PULLBC
                                        RSTPULL
0C8A'   DF              +               RST     018H
0C8B'   E1                              POP     HL
0C8C'   A7                              AND     A
0C8D'   ED 52                           SBC     HL,DE
0C8F'   28 E6                           JR      Z,UCMP          ;ARE HIGH 16 BIT EQUAL ?

0C91'   19                              ADD     HL,DE
0C92'   EB                              EX      DE,HL
0C93'   CD 0C99'                        CALL    GREATER         ;COMPARE ONLY HIGHER 16 BIT
                                        RSTPULL
0C96'   DF              +               RST     018H
0C97'   18 88                           JR      CMPPUSH
                                ;=================================================================
0C99'                           GREATER:
0C99'   7C                              LD      A,H
0C9A'   AA                              XOR     D
0C9B'   FA 0CA0'                        JP      M,GRTRQUIT      ;DIFFERENT SIGNS?

0C9E'   ED 52                           SBC     HL,DE
0CA0'                           GRTRQUIT:
```

```
0CA0'   CB 14                       RL      H               ;SIGN IN C
0CA2'   C9                          RET
                            ;=================================================================
0CA3'   55 AA                       DB      'U','*' OR CLAST
0CA5'   0C82'                       DW      DLT-1
0CA7'   02                          DB      2
0CA8'                       UMUL:
0CA8'   0CAA'                       DW      $+2

                                    RSTPULL
0CAA'   DF              +           RST     018H
0CAB'   CD 084E'                    CALL    PULLBC
0CAE'   21 0000                     LD      HL,0
0CB1'   3E 10                       LD      A,16            ;SET BIT COUNTER
0CB3'                       UMULLOOP:
0CB3'   29                          ADD     HL,HL
0CB4'   EB                          EX      DE,HL
0CB5'   ED 6A                       ADC     HL,HL
0CB7'   EB                          EX      DE,HL
0CB8'   30 04                       JR      NC,UMULNEXT     ;MULTIPLICATOR BIT = 0?
0CBA'   09                          ADD     HL,BC
0CBB'   30 01                       JR      NC,UMULNEXT     ;NO OVERFLOW?
0CBD'   13                          INC     DE
0CBE'                       UMULNEXT:
0CBE'   3D                          DEC     A
0CBF'   20 F2                       JR      NZ,UMULLOOP     ;NOT YET ALL BITS?
0CC1'   EB                          EX      DE,HL
0CC2'   18 2F                       JR      PUSHDEHL
                            ;=================================================================
0CC4'                       DIV32BY16:
0CC4'   0CC6'                       DW      $+2

                                    RSTPULL                 ;DIVISOR
0CC6'   DF              +           RST     018H
0CC7'   D9                          EXX
                                    RSTPULL                 ;DIVIDEND H
0CC8'   DF              +           RST     018H
0CC9'   D5                          PUSH    DE
                                    RSTPULL                 ;DIVIDEND L
0CCA'   DF              +           RST     018H
0CCB'   E1                          POP     HL

0CCC'   7C                          LD      A,H
0CCD'   B5                          OR      L
0CCE'   3E 21                       LD      A,33            ;USUAL BIT COUNTER
0CD0'   20 03                       JR      NZ,D32GOON      ;DIVIDEND > 65535 ?
0CD2'   EB                          EX      DE,HL
0CD3'   3E 11                       LD      A,17            ;SHORT CALCULATION
0CD5'                       D32GOON:
0CD5'   D9                          EXX
0CD6'   47                          LD      B,A
0CD7'   AF                          XOR     A
0CD8'   67                          LD      H,A
0CD9'   6F                          LD      L,A
0CDA'   4F                          LD      C,A             ;PREPARE CALCULATION
0CDB'                       D32LOOP:
```

```
0CDB'   ED 6A                       ADC     HL,HL
0CDD'   9F                          SBC     A,A
0CDE'   A7                          AND     A
0CDF'   ED 52                       SBC     HL,DE           ;TEST SUBTRACTION
0CE1'   99                          SBC     A,C
0CE2'   30 01                       JR      NC,D32NEXT
0CE4'   19                          ADD     HL,DE           ;UNDO SUBTRACTION
0CE5'                       D32NEXT:
0CE5'   3F                          CCF
0CE6'   D9                          EXX
0CE7'   EB                          EX      DE,HL
0CE8'   ED 6A                       ADC     HL,HL
0CEA'   EB                          EX      DE,HL
0CEB'   ED 6A                       ADC     HL,HL
0CED'   D9                          EXX
0CEE'   10 EB                       DJNZ    D32LOOP         ;NOT YET ALL BITS?
0CF0'   EB                          EX      DE,HL
                                    RSTPUSH                 ;SAVE REMAINDER
0CF1'   D7              +           RST     010H
0CF2'   D9                          EXX                     ;GET QUOTIENT

0CF3'                       PUSHDEHL:
0CF3'   E5                          PUSH    HL
                                    RSTPUSH
0CF4'   D7              +           RST     010H
0CF5'   D1                          POP     DE
                                    RSTPUSH
0CF6'   D7              +           RST     010H
0CF7'   FD E9                       JP      (IY)
                            ;=================================================================
0CF9'   2F 4D 4F C4                 DB      '/MO','D' OR CLAST
0CFD'   0CA7'                       DW      UMUL-1
0CFF'   04                          DB      4
0D00'                       DIVMOD:
0D00'   0EC3'                       DW      DOCOL
0D02'   0885' 08D2'                 DW      SWAP,GTR,I,ABS          ;PREPARE DIVIDEND
0D06'   12E9' 0C0D'
0D0A'   104B'                       DW      GETBYTE
0D0C'   00                          DB      0
0D0D'                       DIVMOD2:
0D0D'   08FF' 086B'                 DW      ROT,DUP,I
0D11'   12E9'
0D13'   0E60'                       DW      LXOR                    ;CALCULATE SIGN
0D15'   08D2' 0C0D'                 DW      GTR,ABS                 ;PREPARE DIVISOR
0D19'   0D8C'                       DW      UDIVMOD
0D1B'   08DF' 0D94'                 DW      RGT,IFN0NEG,SWAP        ;SIGN QUOTIENT
0D1F'   0885'
0D21'   08DF' 0D94'                 DW      RGT,IFN0NEG,SWAP        ;SIGN REMAINDER
0D25'   0885'
0D27'   04B6'                       DW      SEMIS
                            ;=================================================================
0D29'   2A 2F 4D 4F                 DB      '*/MO','D' OR CLAST
0D2D'   C4
0D2E'   0CFF'                       DW      DIVMOD-1
0D30'   05                          DB      5
0D31'                       MULDIVMOD:
```

```
0D31'   0EC3'                     DW      DOCOL
0D33'   08FF' 08D2'               DW      ROT,GTR,I,ABS           ;PREPARE 1ST MULTIPLIER
0D37'   12E9' 0C0D'
0D3B'   08FF' 086B'               DW      ROT,DUP,RGT,LXOR        ;CALCULATE SIGN
0D3F'   08DF' 0E60'
0D43'   08D2' 0C0D'               DW      GTR,ABS                 ;PREPARE 2ND MULTIPLIER
0D47'   0CA8'                     DW      UMUL
0D49'   1276' FFC1                DW      DOREPEAT,DIVMOD2-$-1
                          ;=================================================================
0D4D'   AF                        DB      '/' OR CLAST
0D4E'   0D30'                     DW      MULDIVMOD-1
0D50'   01                        DB      1
0D51'                     DIV:
0D51'   0EC3'                     DW      DOCOL
0D53'   0D00'                     DW      DIVMOD
0D55'   0885' 0879'               DW      SWAP,DROP               ;REMOVE REMAINDER
0D59'   04B6'                     DW      SEMIS
                          ;=================================================================
0D5B'   4D 4F C4                  DB      'MO','D' OR CLAST
0D5E'   0D50'                     DW      DIV-1
0D60'   03                        DB      3
0D61'                     MOD:
0D61'   0EC3'                     DW      DOCOL
0D63'   0D00'                     DW      DIVMOD
0D65'   0879'                     DW      DROP                    ;REMOVE QUOTIENT
0D67'   04B6'                     DW      SEMIS
                          ;=================================================================
0D69'   AA                        DB      '*' OR CLAST
0D6A'   0D60'                     DW      MOD-1
0D6C'   01                        DB      1
0D6D'                     MUL:
0D6D'   0EC3'                     DW      DOCOL
0D6F'   0CA8' 0879'               DW      UMUL,DROP               ;REMOVE HIGHER 16-BIT
0D73'   04B6'                     DW      SEMIS
                          ;=================================================================
0D75'   2A AF                     DB      '*','/' OR CLAST
0D77'   0D6C'                     DW      MUL-1
0D79'   02                        DB      2
0D7A'                     MULDIV:
0D7A'   0EC3'                     DW      DOCOL
0D7C'   0D31'                     DW      MULDIVMOD               ;*/MOD
0D7E'   0885' 0879'               DW      SWAP,DROP               ;REMOVE REMAINDER
0D82'   04B6'                     DW      SEMIS
                          ;=================================================================
0D84'   55 2F 4D 4F               DB      'U/MO','D' OR CLAST
0D88'   C4
0D89'   0D79'                     DW      MULDIV-1
0D8B'   05                        DB      5
0D8C'                     UDIVMOD:
0D8C'   0EC3'                     DW      DOCOL
0D8E'   0CC4' 0879'               DW      DIV32BY16,DROP
0D92'   04B6'                     DW      SEMIS
                          ;=================================================================
0D94'                     IFN0NEG:
0D94'   0EC3'                     DW      DOCOL
0D96'   0C2E' 1283'               DW      ZEROLT,DOIF,I0NEND-$-1
```

```
0D9A'   0003
0D9C'   0DA9'                      DW      NEGATE                  ;SAME SIGN AS TOS
0D9E'                      I0NEND:
0D9E'   04B6'                      DW      SEMIS
                           ;=================================================================
0DA0'   4E 45 47 41                DB      'NEGAT','E' OR CLAST
0DA4'   54 C5
0DA6'   0D8B'                      DW      UDIVMOD-1
0DA8'   06                         DB      6
0DA9'                      NEGATE:
0DA9'   0DAB'                      DW      $+2

0DAB'   01 0002                    LD      BC,2            ;2 BYTES
0DAE'   18 0F                      JR      DONEGATE
                           ;=================================================================
0DB0'   44 4E 45 47                DB      'DNEGAT','E' OR CLAST
0DB4'   41 54 C5
0DB7'   0DA8'                      DW      NEGATE-1
0DB9'   07                         DB      7
0DBA'                      DNEGATE:
0DBA'   0DBC'                      DW      $+2

0DBC'   01 0004                    LD      BC,4            ;4 BYTES
0DBF'                      DONEGATE:
0DBF'   2A 3C3B                    LD      HL,(SPARE)
0DC2'   A7                         AND     A
0DC3'   ED 42                      SBC     HL,BC           ;POINTER TO NUMBER IN DATA STACK
0DC5'                      DNLOOP:
0DC5'   78                         LD      A,B             ;LOAD 0 WITHOUT CLEARING CARRY
0DC6'   9E                         SBC     A,(HL)
0DC7'   77                         LD      (HL),A          ;NEGATE BYTE
0DC8'   23                         INC     HL
0DC9'   0D                         DEC     C
0DCA'   20 F9                      JR      NZ,DNLOOP       ;NOT YET ALL BYTES?
0DCC'   FD E9                      JP      (IY)
                           ;=================================================================
0DCE'   AB                         DB      '+' OR CLAST
0DCF'   0DB9'                      DW      DNEGATE-1
0DD1'   01                         DB      1
0DD2'                      PLUS:
0DD2'   0DD4'                      DW      $+2

                                   RSTPULL
0DD4'   DF              +          RST     018H
0DD5'   D5                         PUSH    DE
                                   RSTPULL
0DD6'   DF              +          RST     018H
0DD7'   E1                         POP     HL
0DD8'   19                         ADD     HL,DE
0DD9'   EB                         EX      DE,HL
                                   RSTPUSH
0DDA'   D7              +          RST     010H
0DDB'   FD E9                      JP      (IY)
                           ;=================================================================
0DDD'   AD                         DB      '-' OR CLAST
0DDE'   0DD1'                      DW      PLUS-1
```

```
0DE0'   01                          DB      1
0DE1'                       MINUS:
0DE1'   0EC3'                       DW      DOCOL
0DE3'   0DA9' 0DD2'                 DW      NEGATE,PLUS
0DE7'   04B6'                       DW      SEMIS
                            ;=================================================================
0DE9'   44 AB                       DB      'D','+' OR CLAST
0DEB'   0DE0'                       DW      MINUS-1
0DED'   02                          DB      2
0DEE'                       DPLUS:
0DEE'   0DF0'                       DW      $+2

                                    RSTPULL
0DF0'   DF              +           RST     018H
0DF1'   D5                          PUSH    DE
0DF2'   CD 084E'                    CALL    PULLBC
                                    RSTPULL
0DF5'   DF              +           RST     018H
0DF6'   D5                          PUSH    DE
                                    RSTPULL
0DF7'   DF              +           RST     018H
0DF8'   EB                          EX      DE,HL
0DF9'   09                          ADD     HL,BC
0DFA'   EB                          EX      DE,HL
                                    RSTPUSH
0DFB'   D7              +           RST     010H
0DFC'   C1                          POP     BC
0DFD'   E1                          POP     HL
0DFE'   ED 4A                       ADC     HL,BC
0E00'   EB                          EX      DE,HL
                                    RSTPUSH
0E01'   D7              +           RST     010H
0E02'   FD E9                       JP      (IY)
                            ;=================================================================
0E04'   31 AB                       DB      '1','+' OR CLAST
0E06'   0DED'                       DW      DPLUS-1
0E08'   02                          DB      2
0E09'                       ONEPLUS:
0E09'   0E0B'                       DW      $+2

                                    RSTPULL
0E0B'   DF              +           RST     018H
0E0C'   18 09                       JR      XPLUS
                            ;=================================================================
0E0E'   32 AB                       DB      '2','+' OR CLAST
0E10'   0E08'                       DW      ONEPLUS-1
0E12'   02                          DB      2
0E13'                       TWOPLUS:
0E13'   0E15'                       DW      $+2

                                    RSTPULL
0E15'   DF              +           RST     018H
0E16'   13                          INC     DE
0E17'                       XPLUS:
0E17'   13                          INC     DE
0E18'   18 14                       JR      XPLUSMINUS
```

```
                                ;=================================================================
0E1A'   31 AD                           DB      '1','-' OR CLAST
0E1C'   0E12'                           DW      TWOPLUS-1
0E1E'   02                              DB      2
0E1F'                           ONEMINUS:
0E1F'   0E21'                           DW      $+2

                                        RSTPULL
0E21'   DF              +               RST     018H
0E22'   18 09                           JR      XMINUS
                                ;=================================================================
0E24'   32 AD                           DB      '2','-' OR CLAST
0E26'   0E1E'                           DW      ONEMINUS-1
0E28'   02                              DB      2
0E29'                           TWOMINUS:
0E29'   0E2B'                           DW      $+2

                                        RSTPULL
0E2B'   DF              +               RST     018H
0E2C'   1B                              DEC     DE
0E2D'                           XMINUS:
0E2D'   1B                              DEC     DE
0E2E'                           XPLUSMINUS:
                                        RSTPUSH
0E2E'   D7              +               RST     010H
0E2F'   FD E9                           JP      (IY)
                                ;=================================================================
0E31'   4F D2                           DB      'O','R' OR CLAST
0E33'   0E28'                           DW      TWOMINUS-1
0E35'   02                              DB      2
0E36'                           LOR:
0E36'   0E38'                           DW      $+2

                                        RSTPULL
0E38'   DF              +               RST     018H
0E39'   CD 084E'                        CALL    PULLBC
0E3C'   7B                              LD      A,E
0E3D'   B1                              OR      C
0E3E'   5F                              LD      E,A
0E3F'   7A                              LD      A,D
0E40'   B0                              OR      B
0E41'   57                              LD      D,A
                                        RSTPUSH
0E42'   D7              +               RST     010H
0E43'   FD E9                           JP      (IY)
                                ;=================================================================
0E45'   41 4E C4                        DB      'AN','D' OR CLAST
0E48'   0E35'                           DW      LOR-1
0E4A'   03                              DB      3
0E4B'                           LAND:
0E4B'   0E4D'                           DW      $+2

                                        RSTPULL
0E4D'   DF              +               RST     018H
0E4E'   CD 084E'                        CALL    PULLBC
0E51'   7B                              LD      A,E
```

```
0E52'   A1                          AND     C
0E53'   5F                          LD      E,A
0E54'   7A                          LD      A,D
0E55'   A0                          AND     B
0E56'   57                          LD      D,A
                                    RSTPUSH
0E57'   D7              +           RST     010H
0E58'   FD E9                       JP      (IY)
                            ;================================================================
0E5A'   58 4F D2                    DB      'XO','R' OR CLAST
0E5D'   0E4A'                       DW      LAND-1
0E5F'   03                          DB      3
0E60'                       LXOR:
0E60'   0E62'                       DW      $+2

                                    RSTPULL
0E62'   DF              +           RST     018H
0E63'   CD 084E'                    CALL    PULLBC
0E66'   7B                          LD      A,E
0E67'   A9                          XOR     C
0E68'   5F                          LD      E,A
0E69'   7A                          LD      A,D
0E6A'   A8                          XOR     B
0E6B'   57                          LD      D,A
                                    RSTPUSH
0E6C'   D7              +           RST     010H
0E6D'   FD E9                       JP      (IY)
                            ;================================================================
0E6F'   4D 41 D8                    DB      'MA','X' OR CLAST
0E72'   0E5F'                       DW      LXOR-1
0E74'   03                          DB      3
0E75'                       MAX:
0E75'   0EC3'                       DW      DOCOL
0E77'   0912' 0912'                 DW      OVER,OVER,LT            ;COMPARE NUMBERS
0E7B'   0C65'
0E7D'   1271' 000F                  DW      DOELSE,MINMAX-$-1
                            ;================================================================
0E81'   4D 49 CE                    DB      'MI','N' OR CLAST
0E84'   0E74'                       DW      MAX-1
0E86'   03                          DB      3
0E87'                       MIN:
0E87'   0EC3'                       DW      DOCOL
0E89'   0912' 0912'                 DW      OVER,OVER,GT            ;COMPARE NUMBERS
0E8D'   0C56'
0E8F'                       MINMAX:
0E8F'   1283' 0003                  DW      DOIF,MINMAXEND-$-1
0E93'   0885'                       DW      SWAP                    ;SWAP IF NEEDED
0E95'                       MINMAXEND:
0E95'   0879'                       DW      DROP                    ;REMOVE THE OTHER NUMBER
0E97'   04B6'                       DW      SEMIS
                            ;================================================================
0E99'   44 45 43 49                 DB      'DECIMA','L' OR CLAST
0E9D'   4D 41 CC
0EA0'   0E86'                       DW      MIN-1
0EA2'   07                          DB      7
0EA3'                       DECIMAL:
```

```
0EA3'   0EA5'                       DW      $+2

0EA5'   DD 36 3F 0A                 LD      (IX+VBASE-MEMBEG),10
0EA9'   FD E9                       JP      (IY)
                            ;=================================================================
0EAB'                       NCOLON:
0EAB'   BA                          DB      ':' OR CLAST
0EAC'   0EA2'                       DW      DECIMAL-1
0EAE'   01                          DB      1
0EAF'                       COLON:
0EAF'   1085' 0EC3'                 DW      DODEFINER,DOCOL
0EB3'   104B'                       DW      GETBYTE
0EB5'   0A                          DB      10                      ;SET CHECK VALUE
0EB6'   1A0E'                       DW      SEMICODE

0EB8'   21 3C3E                     LD      HL,FLAGS
0EBB'   7E                          LD      A,(HL)
0EBC'   F6 44                       OR      (1 SHL 6) OR (1 SHL 2)
0EBE'   77                          LD      (HL),A          ;TURN ON COMPILER
0EBF'   FD E9                       JP      (IY)
                            ;=================================================================
0EC1'   FFE9                        DW      NCOLON-$-1
0EC3'                       DOCOL:
0EC3'   EB                          EX      DE,HL           ;NEW INSTRUCTION TO STACK
0EC4'   C3 04BA'                    JP      NEXTSUB
                            ;=================================================================
0EC7'                       NCREATE:
0EC7'   43 52 45 41                 DB      'CREAT','E' OR CLAST
0ECB'   54 C5
0ECD'   0EAE'                       DW      COLON-1
0ECF'   06                          DB      6
0ED0'                       CREATE:
0ED0'   0EC3'                       DW      DOCOL
0ED2'   104B'                       DW      GETBYTE
0ED4'   20                          DB      ' '
0ED5'   05AB' 0EFB'                 DW      WORD,CRHEADER           ;PREPARE HEADER
0ED9'   0688' 0F4E'                 DW      ZERO,KOMMA
0EDD'   0480' 08B3'                 DW      CURRENT,AT
0EE1'   086B' 08B3'                 DW      DUP,AT,KOMMA            ;CREATE LINK
0EE5'   0F4E'
0EE7'   0460' 0885'                 DW      HERE,SWAP,EXCLAM        ;STORE ADDRESS
0EEB'   08C1'
0EED'   0499' 0896'                 DW      PAD,CAT,CKOMMA
0EF1'   0F5F'
0EF3'   1011' 0FEC'                 DW      GETWORD,DOCREATE,KOMMA  ;CREATE CODE-FIELD
0EF7'   0F4E'
0EF9'   04B6'                       DW      SEMIS
                            ;=================================================================
0EFB'                       CRHEADER:
0EFB'   0EFD'                       DW      $+2

0EFD'   CD 0F2E'                    CALL    LINKHERE

                                    RSTPULL
0F00'   DF              +           RST     018H
0F01'   1A                          LD      A,(DE)          ;GET NAME LENGTH
```

```
0F02'   3D                              DEC     A
0F03'   FE 3F                           CP      03FH
0F05'   38 02                           JR      C,CHGOON        ;NAME NOT TOO LONG?
                                        RSTERR  ERRNAME
0F07'   E7              +               RST     020H
0F08'   06              +               DB      ERRNAME

0F09'                           CHGOON:
0F09'   C6 08                           ADD     A,8             ;(+1) + SIZE,LINK,NAME#,CODEWORD
0F0B'   4F                              LD      C,A
0F0C'   06 00                           LD      B,0
0F0E'   CD 0F8C'                        CALL    MEMCHECK

0F11'   1A                              LD      A,(DE)
0F12'   4F                              LD      C,A
0F13'   2A 3C37                         LD      HL,(STKBOT)
0F16'   D5                              PUSH    DE
0F17'   CD 0F9E'                        CALL    ALLOC           ;RESERVE MEMORY
0F1A'   D1                              POP     DE

0F1B'   1A                              LD      A,(DE)
0F1C'   47                              LD      B,A             ;CHARS COUNT
0F1D'                           CHLOOP:
0F1D'   13                              INC     DE
0F1E'   1A                              LD      A,(DE)
0F1F'   CD 0807'                        CALL    TOUPPER
0F22'   77                              LD      (HL),A
0F23'   23                              INC     HL
0F24'   10 F7                           DJNZ    CHLOOP          ;STORE NAME

0F26'   22 3C39                         LD      (DICT),HL
0F29'   2B                              DEC     HL
0F2A'   CB FE                           SET     7,(HL)          ;TAG NAME-END
0F2C'   FD E9                           JP      (IY)
                                ;=================================================================
0F2E'                           LINKHERE:
0F2E'   DD CB 3E 56                     BIT     2,(IX+FLAGS-MEMBEG)
0F32'   28 02                           JR      Z,LHGOON        ;NOT COMPILE-MODE ?
                                        RSTERR  ERRMODE
0F34'   E7              +               RST     020H
0F35'   0C              +               DB      ERRMODE

0F36'                           LHGOON:
0F36'   2A 3C37                         LD      HL,(STKBOT)
0F39'   ED 5B 3C39                      LD      DE,(DICT)
0F3D'   AF                              XOR     A
0F3E'   ED 52                           SBC     HL,DE
0F40'   EB                              EX      DE,HL
0F41'   73                              LD      (HL),E
0F42'   23                              INC     HL
0F43'   72                              LD      (HL),D          ;CREATE LINK

0F44'   67                              LD      H,A
0F45'   6F                              LD      L,A
0F46'   22 3C39                         LD      (DICT),HL
0F49'   C9                              RET
```

```
                                        ;==================================================================
0F4A'   AC                                      DB      ',' OR CLAST
0F4B'   0ECF'                                   DW      CREATE-1
0F4D'   01                                      DB      1
0F4E'                                   KOMMA:
0F4E'   0EC3'                                   DW      DOCOL
0F50'   0F83' 0460'                             DW      ALLOT2,HERE,TWOMINUS,EXCLAM
0F54'   0E29' 08C1'
0F58'   04B6'                                   DW      SEMIS
                                        ;==================================================================
0F5A'   43 AC                                   DB      'C',',' OR CLAST
0F5C'   0F4D'                                   DW      KOMMA-1
0F5E'   02                                      DB      2
0F5F'                                   CKOMMA:
0F5F'   0EC3'                                   DW      DOCOL
0F61'   104B'                                   DW      GETBYTE
0F63'   01                                      DB      1
0F64'   0F76' 0460'                             DW      ALLOT,HERE,ONEMINUS,CEXCLAM
0F68'   0E1F' 08A5'
0F6C'   04B6'                                   DW      SEMIS
                                        ;==================================================================
0F6E'   41 4C 4C 4F                             DB      'ALLO','T' OR CLAST
0F72'   D4
0F73'   0F5E'                                   DW      CKOMMA-1
0F75'   05                                      DB      5
0F76'                                   ALLOT:
0F76'   0F78'                                   DW      $+2

0F78'   CD 084E'                                CALL    PULLBC
0F7B'   2A 3C37                                 LD      HL,(STKBOT)
0F7E'   CD 0F9E'                                CALL    ALLOC
0F81'   FD E9                                   JP      (IY)
                                        ;==================================================================
0F83'                                   ALLOT2:
0F83'   0EC3'                                   DW      DOCOL
0F85'   104B'                                   DW      GETBYTE
0F87'   02                                      DB      2
0F88'   0F76'                                   DW      ALLOT
0F8A'   04B6'                                   DW      SEMIS
                                        ;==================================================================
0F8C'                                   MEMCHECK:
0F8C'   21 001E                                 LD      HL,30
0F8F'                                   MEMCHECK2:
0F8F'   C5                                      PUSH    BC
0F90'   09                                      ADD     HL,BC
0F91'   ED 4B 3C3B                              LD      BC,(SPARE)
0F95'   09                                      ADD     HL,BC           ;NEW END ADDRESS
0F96'   C1                                      POP     BC
0F97'   38 03                                   JR      C,MCERROR       ;MEMORY OVERFLOW?
0F99'   ED 72                                   SBC     HL,SP
0F9B'   D8                                      RET     C               ;NO COLLISION WITH STACK
0F9C'                                   MCERROR:
                                                RSTERR  ERRMEM
0F9C'   E7                      +               RST     020H
0F9D'   01                      +               DB      ERRMEM
                                        ;==================================================================
```

```
0F9E'                           ALLOC:
0F9E'   EB                              EX      DE,HL
0F9F'   21 0028                         LD      HL,40
0FA2'   CD 0F8F'                        CALL    MEMCHECK2       ;CHECKING A BIT MORE

0FA5'   2A 3C37                         LD      HL,(STKBOT)
0FA8'   09                              ADD     HL,BC
0FA9'   22 3C37                         LD      (STKBOT),HL
0FAC'   2A 3C3B                         LD      HL,(SPARE)
0FAF'   E5                              PUSH    HL
0FB0'   09                              ADD     HL,BC
0FB1'   22 3C3B                         LD      (SPARE),HL      ;ADVANCE POINTER

0FB4'   E3                              EX      (SP),HL
0FB5'   E5                              PUSH    HL
0FB6'   A7                              AND     A
0FB7'   ED 52                           SBC     HL,DE
0FB9'   44                              LD      B,H
0FBA'   4D                              LD      C,L             ;DISTANCE = OLD SPACE - DE
0FBB'   E1                              POP     HL
0FBC'   D1                              POP     DE
0FBD'   C8                              RET     Z               ;NOTHING TO MOVE?

0FBE'   2B                              DEC     HL
0FBF'   1B                              DEC     DE
0FC0'   ED B8                           LDDR
0FC2'   23                              INC     HL              ;MOVE PARAMETER STACK
0FC3'   C9                              RET
                                ;=================================================================
0FC4'                           NVARIABLE:
0FC4'   56 41 52 49                     DB      'VARIABL','E' OR CLAST
0FC8'   41 42 4C C5
0FCC'   0F75'                           DW      ALLOT-1
0FCE'   08                              DB      8
0FCF'                           VARIABLE:
0FCF'   1085' 0FF0'                     DW      DODEFINER,DOVARIABLE
0FD3'   0F4E'                           DW      KOMMA
0FD5'   04B6'                           DW      SEMIS
                                ;=================================================================
0FD7'                           NCONSTANT:
0FD7'   43 4F 4E 53                     DB      'CONSTAN','T' OR CLAST
0FDB'   54 41 4E D4
0FDF'   0FCE'                           DW      VARIABLE-1
0FE1'   08                              DB      8
0FE2'                           CONSTANT:
0FE2'   1085' 0FF5'                     DW      DODEFINER,DOCONSTANT
0FE6'   0F4E'                           DW      KOMMA
0FE8'   04B6'                           DW      SEMIS
                                ;=================================================================
0FEA'   FEDC                            DW      NCREATE-$-1
0FEC'                           DOCREATE:
0FEC'   18 02                           JR      DOVARIABLE
                                ;=================================================================
0FEE'   FFD5                            DW      NVARIABLE-$-1
0FF0'                           DOVARIABLE:
                                        RSTPUSH
```

```
0FF0'   D7              +          RST     010H
0FF1'   FD E9                      JP      (IY)
                           ;=================================================================
0FF3'   FFE3                       DW      NCONSTANT-$-1
0FF5'                      DOCONSTANT:
0FF5'   EB                         EX      DE,HL
0FF6'   5E                         LD      E,(HL)
0FF7'   23                         INC     HL
0FF8'   56                         LD      D,(HL)
                                   RSTPUSH                 ;VALUE ON STACK
0FF9'   D7              +          RST     010H
0FFA'   FD E9                      JP      (IY)
                           ;=================================================================
0FFC'   4C 49 54 45                DB      'LITERA','L' OR CLAST
1000'   52 41 CC
1003'   0FE1'                      DW      CONSTANT-1
1005'   47                         DB      7 OR IMM
1006'                      LITERAL:
1006'   1108' 1011'                DW      DOCOMPILER,GETWORD
100A'   0F4E'                      DW      KOMMA
100C'   04B6'                      DW      SEMIS
                           ;=================================================================
100E'   02                         DB      2
100F'   FFFF                       DW      -1
1011'                      GETWORD:
1011'   1013'                      DW      $+2

1013'   06 01                      LD      B,1             ;ONLY ONE WORD
1015'                      GWLOOP:
1015'   E1                         POP     HL
1016'   5E                         LD      E,(HL)
1017'   23                         INC     HL
1018'   56                         LD      D,(HL)          ;GET WORD
1019'                      GWGOON:
1019'   23                         INC     HL
101A'   E5                         PUSH    HL
                                   RSTPUSH                 ;WORD ON STACK
101B'   D7              +          RST     010H
101C'   10 F7                      DJNZ    GWLOOP
101E'                      GWQUIT:
101E'   FD E9                      JP      (IY)
                           ;=================================================================
1020'                      NASCII:
1020'   41 53 43 49                DB      'ASCI','I' OR CLAST
1024'   C9
1025'   1005'                      DW      LITERAL-1
1027'   45                         DB      5 OR IMM
1028'                      ASCII:
1028'   0EC3'                      DW      DOCOL
102A'   104B'                      DW      GETBYTE
102C'   20                         DB      ' '
102D'   05AB' 0E09'                DW      WORD,ONEPLUS,CAT
1031'   0896'
1033'   1A0E'                      DW      SEMICODE

1035'   DD CB 3E 76                BIT     6,(IX+FLAGS-MEMBEG)
```

```
1039'   28 E3                       JR      Z,GWQUIT        ;COMPILER OFF?

103B'   CD 04B9'                    CALL    NEXT
103E'   1011' 104B'                 DW      GETWORD,GETBYTE,KOMMA
1042'   0F4E'
1044'   0F5F'                       DW      CKOMMA
1046'   04B6'                       DW      SEMIS
                            ;=================================================================
1048'   01                          DB      1
1049'   FFD6                        DW      NASCII-$-1
104B'                       GETBYTE:
104B'   104D'                       DW      $+2

104D'   E1                          POP     HL
104E'   5E                          LD      E,(HL)
104F'   16 00                       LD      D,0
1051'   06 01                       LD      B,1
1053'   18 C4                       JR      GWGOON
                            ;=================================================================
1055'                       LITFLOAT:
1055'   1108' 1064'                 DW      DOCOMPILER,GETFLOAT
1059'   0885' 0F4E'                 DW      SWAP,KOMMA,KOMMA
105D'   0F4E'
105F'   04B6'                       DW      SEMIS
                            ;=================================================================
1061'   04                          DB      4
1062'   FFFF                        DW      -1
1064'                       GETFLOAT:
1064'   1066'                       DW      $+2

1066'   06 02                       LD      B,2
1068'   18 AB                       JR      GWLOOP
                            ;=================================================================
106A'                       NDEFINER:
106A'   44 45 46 49                 DB      'DEFINE','R' OR CLAST
106E'   4E 45 D2
1071'   1027'                       DW      ASCII-1
1073'   07                          DB      7
1074'                       DEFINER:
1074'   1085' 1085'                 DW      DODEFINER,DODEFINER
1078'   0460' 104B'                 DW      HERE,GETBYTE
107C'   0C                          DB      12
107D'   0F83'                       DW      ALLOT2
107F'   1276' FE34'                 DW      DOREPEAT,0EB6H-$-1
                            ;=================================================================
1083'   FFE6                        DW      NDEFINER-$-1
1085'                       DODEFINER:
1085'   CD 0FF0'                    CALL    DOVARIABLE
1088'   0ED0'                       DW      CREATE                  ;CREATE HEADER
108A'   086B' 08B3'                 DW      DUP,AT
108E'   0460' 0E29'                 DW      HERE,TWOMINUS,EXCLAM    ;MAKE LINK
1092'   08C1'
1094'   0E13' 109A'                 DW      TWOPLUS,DROPGOON
1098'   04B6'                       DW      SEMIS
                            ;-----------------------------------------------------------------
109A'                       DROPGOON:
```

```
109A'   109C'                   DW      $+2

                                RSTPULL
109C'   DF              +       RST     018H
109D'   C3 0EC3'                JP      DOCOL
                        ;=================================================================
10A0'   43 41 4C CC             DB      'CAL','L' OR CLAST
10A4'   1073'                   DW      DEFINER-1
10A6'   04                      DB      4
10A7'                   CALL:
10A7'   10A9'                   DW      $+2

                                RSTPULL                 ;GET DESTINATION ADDRESS
10A9'   DF              +       RST     018H
10AA'   EB                      EX      DE,HL
10AB'   E9                      JP      (HL)
                        ;=================================================================
10AC'                   NDOESGT:
10AC'   44 4F 45 53             DB      'DOES','>' OR CLAST
10B0'   BE
10B1'   10F4'                   DW      COMPILER-1
10B3'   45                      DB      5 OR IMM
10B4'                   DOESGT:
10B4'   1108' 10E8'             DW      DOCOMPILER,DODOESGT
10B8'   12D8'                   DW      ASSERT
10BA'   0C                      DB      12                      ;TEST CHECK VALUE
10BB'   10CD'                   DW      DOESPATCH
10BD'   104B'                   DW      GETBYTE
10BF'   CD                      DB      0CDH
10C0'   0F5F'                   DW      CKOMMA
10C2'   1011' 0FF0'             DW      GETWORD,DOVARIABLE,KOMMA;"CALL DOVARIABLE"
10C6'   0F4E'
10C8'   104B'                   DW      GETBYTE
10CA'   0A                      DB      10                      ;SET CHECK VALUE
10CB'   04B6'                   DW      SEMIS
                        ;=================================================================
10CD'                   DOESPATCH:
10CD'   0EC3'                   DW      DOCOL
10CF'   086B' 0E29'             DW      DUP,TWOMINUS,NFA
10D3'   15B5'
10D5'   0460' 0DE1'             DW      HERE,MINUS,ONEMINUS,KOMMA
10D9'   0E1F' 0F4E'
10DD'   0460' 0885'             DW      HERE,SWAP,EXCLAM        ;ADJUST LINK
10E1'   08C1'
10E3'   04B6'                   DW      SEMIS
                        ;=================================================================
10E5'   05                      DB      5
10E6'   FFC5                    DW      NDOESGT-$-1
10E8'                   DODOESGT:
10E8'   04B8'                   DW      RSEMIS
                        ;=================================================================
10EA'                   NCOMPILER:
10EA'   43 4F 4D 50             DB      'COMPILE','R' OR CLAST
10EE'   49 4C 45 D2
10F2'   10A6'                   DW      CALL-1
10F4'   08                      DB      8
```

```
10F5'                           COMPILER:
10F5'   1085' 1108'                     DW      DODEFINER,DOCOMPILER
10F9'   1160'                           DW      IMMEDIATE
10FB'   0460'                           DW      HERE
10FD'   104B'                           DW      GETBYTE
10FF'   0B                              DB      11
1100'   0F83'                           DW      ALLOT2
1102'   1276' FDB1'                     DW      DOREPEAT,0EB6H-$-1
                                ;=================================================================
1106'   FFE3                            DW      NCOMPILER-$-1
1108'                           DOCOMPILER:
1108'   DD CB 3E 76                     BIT     6,(IX+FLAGS-MEMBEG)
110C'   20 02                           JR      NZ,DOCOMGOON    ;COMPILER ON?

                                        RSTERR  ERRIMM
110E'   E7              +               RST     020H
110F'   04              +               DB      ERRIMM

1110'                           DOCOMGOON:
1110'   CD 0FF0'                        CALL    DOVARIABLE
1113'   086B' 08B3'                     DW      DUP,AT,KOMMA
1117'   0F4E'
1119'   1276' FF78'                     DW      DOREPEAT,1094H-$-1
                                ;=================================================================
111D'                           NRUNSGT:
111D'   52 55 4E 53                     DB      'RUNS','>' OR CLAST
1121'   BE
1122'   10B3'                           DW      DOESGT-1
1124'   45                              DB      5 OR IMM
1125'                           RUNSGT:
1125'   1108' 1140'                     DW      DOCOMPILER,DORUNSGT
1129'   12D8'                           DW      ASSERT
112B'   0B                              DB      11                      ;TEST CHECK VALUE
112C'   0885' 0F5F'                     DW      SWAP,CKOMMA
1130'   10CD'                           DW      DOESPATCH
1132'   1011' 1142'                     DW      GETWORD,RUNSCORR,KOMMA
1136'   0F4E'
1138'   104B'                           DW      GETBYTE
113A'   0A                              DB      10                      ;SET CHECK VALUE
113B'   04B6'                           DW      SEMIS
                                ;-----------------------------------------------------------------
113D'   05                              DB      5
113E'   FFDE                            DW      NRUNSGT-$-1
1140'                           DORUNSGT:
1140'   04B8'                           DW      RSEMIS
                                ;-----------------------------------------------------------------
1142'                           RUNSCORR:
1142'   E1                              POP     HL
1143'   D5                              PUSH    DE
1144'   EB                              EX      DE,HL
                                        RSTPUSH
1145'   D7              +               RST     010H
1146'   42                              LD      B,D
1147'   4B                              LD      C,E
1148'   D1                              POP     DE
1149'   D5                              PUSH    DE
```

```
114A'   1B                              DEC     DE
114B'   1B                              DEC     DE
114C'   CD 159E'                        CALL    SKIPOFFS        ;NEXT FORTH ADDRESS
114F'   D1                              POP     DE
1150'   C5                              PUSH    BC
1151'   C3 0EC3'                        JP      DOCOL
                                ;=================================================================
1154'   49 4D 4D 45                     DB      'IMMEDIAT','E' OR CLAST
1158'   44 49 41 54
115C'   C5
115D'   1124'                           DW      RUNSGT-1
115F'   09                              DB      9
1160'                           IMMEDIATE:
1160'   0EC3'                           DW      DOCOL
1162'   0480' 08B3'                     DW      CURRENT,AT,AT
1166'   08B3'
1168'   1A0E'                           DW      SEMICODE
                                        RSTPULL
116A'   DF                    +         RST     018H
116B'   EB                              EX      DE,HL
116C'   CB F6                           SET     6,(HL)          ;SET IMMEDIATE BIT
116E'   FD E9                           JP      (IY)
                                ;=================================================================
1170'   56 4F 43 41                     DB      'VOCABULAR','Y' OR CLAST
1174'   42 55 4C 41
1178'   52 D9
117A'   115F'                           DW      IMMEDIATE-1
117C'   0A                              DB      10
117D'                           VOCABULARY:
117D'   1085' 11B5'                     DW      DODEFINER,SETCONTEXT
1181'   0480' 08B3'                     DW      CURRENT,AT
1185'   0E13' 0F4E'                     DW      TWOPLUS,KOMMA
1189'   0688' 0F5F'                     DW      ZERO,CKOMMA             ;PREPARE LINK
118D'   0460' 1011'                     DW      HERE,GETWORD,VOCLNK
1191'   3C35
1193'   086B' 08B3'                     DW      DUP,AT,KOMMA,EXCLAM     ;TOGGLE COMPILER
1197'   0F4E' 08C1'
119B'   04B6'                           DW      SEMIS
                                ;=================================================================
119D'   44 45 46 49                     DB      'DEFINITION','S' OR CLAST
11A1'   4E 49 54 49
11A5'   4F 4E D3
11A8'   117C'                           DW      VOCABULARY-1
11AA'   0B                              DB      11
11AB'                           DEFINITIONS:
11AB'   11AD'                           DW      $+2

11AD'   2A 3C33                         LD      HL,(VCONTEXT)
11B0'   22 3C31                         LD      (VCURRENT),HL
11B3'   FD E9                           JP      (IY)
                                ;-----------------------------------------------------------------
11B5'                           SETCONTEXT:
11B5'   ED 53 3C33                      LD      (VCONTEXT),DE
11B9'   FD E9                           JP      (IY)
                                ;=================================================================
11BB'                           NIF:
```

```
11BB'   49 C6                   DB      'I','F' OR CLAST
11BD'   13E0'                   DW      RSQRBR-1
11BF'   42                      DB      2 OR IMM
11C0'                   IF:
11C0'   1108' 1283'             DW      DOCOMPILER,DOIF
11C4'   0460' 104B'             DW      HERE,GETBYTE
11C8'   02                      DB      2
11C9'   0F83'                   DW      ALLOT2
11CB'   04B6'                   DW      SEMIS
                        ;============================================================
11CD'                   NWHILE:
11CD'   57 48 49 4C             DB      'WHIL','E' OR CLAST
11D1'   C5
11D2'   11BF'                   DW      IF-1
11D4'   45                      DB      5 OR IMM
11D5'                   WHILE:
11D5'   1108' 1288'             DW      DOCOMPILER,DOWHILE
11D9'   12D8'                   DW      ASSERT
11DB'   01                      DB      1                       ;TEST CHECK VALUE
11DC'   0460' 104B'             DW      HERE,GETBYTE
11E0'   04                      DB      4
11E1'   0F83'                   DW      ALLOT2
11E3'   04B6'                   DW      SEMIS
                        ;============================================================
11E5'                   NELSE:
11E5'   45 4C 53 C5             DB      'ELS','E' OR CLAST
11E9'   11D4'                   DW      WHILE-1
11EB'   44                      DB      4 OR IMM
11EC'                   ELSE:
11EC'   1108' 1271'             DW      DOCOMPILER,DOELSE
11F0'   12D8'                   DW      ASSERT
11F2'   02                      DB      2                       ;TEST CHECK VALUE
11F3'   0F83'                   DW      ALLOT2
11F5'   1225'                   DW      DOFPATCH
11F7'   0460' 0E29'             DW      HERE,TWOMINUS
11FB'   104B'                   DW      GETBYTE
11FD'   02                      DB      2                       ;SET CHECK VALUE
11FE'   04B6'                   DW      SEMIS
                        ;============================================================
1200'                   NTHEN:
1200'   54 48 45 CE             DB      'THE','N' OR CLAST
1204'   11EB'                   DW      ELSE-1
1206'   44                      DB      4 OR IMM
1207'                   THEN:
1207'   1108' 12A4'             DW      DOCOMPILER,DOTHEN
120B'   12D8'                   DW      ASSERT
120D'   02                      DB      2                       ;TEST CHECK VALUE
120E'   1225'                   DW      DOFPATCH
1210'   04B6'                   DW      SEMIS
                        ;============================================================
1212'                   NBEGIN:
1212'   42 45 47 49             DB      'BEGI','N' OR CLAST
1216'   CE
1217'   1206'                   DW      THEN-1
1219'   45                      DB      5 OR IMM
121A'                   BEGIN:
```

```
121A'   1108' 129F'                 DW      DOCOMPILER,DOBEGIN
121E'   0460'                       DW      HERE
1220'   104B'                       DW      GETBYTE
1222'   01                          DB      1                       ;SET CHECK VALUE
1223'   04B6'                       DW      SEMIS
                            ;=================================================================
1225'                       DOFPATCH:
1225'   0EC3'                       DW      DOCOL
1227'   086B' 0460'                 DW      DUP,HERE,SWAP,MINUS
122B'   0885' 0DE1'
122F'   0E1F' 0885'                 DW      ONEMINUS,SWAP,EXCLAM    ;PATCH JUMP ADDRESS
1233'   08C1'
1235'   04B6'                       DW      SEMIS
                            ;=================================================================
1237'                       DORPATCH:
1237'   0EC3'                       DW      DOCOL
1239'   0460' 0DE1'                 DW      HERE,MINUS,ONEMINUS
123D'   0E1F'
123F'   0F4E'                       DW      KOMMA                   ;PATCH JUMP ADDRESS
1241'   04B6'                       DW      SEMIS
                            ;=================================================================
1243'                       NREPEAT:
1243'   52 45 50 45                 DB      'REPEA','T' OR CLAST
1247'   41 D4
1249'   1219'                       DW      BEGIN-1
124B'   46                          DB      6 OR IMM
124C'                       REPEAT:
124C'   1108' 1276'                 DW      DOCOMPILER,DOREPEAT
1250'   12D8'                       DW      ASSERT
1252'   04                          DB      4                       ;PRUEFWERT TESTEN
1253'   0885'                       DW      SWAP
1255'   1237'                       DW      DORPATCH
1257'   1225'                       DW      DOFPATCH
1259'   04B6'                       DW      SEMIS
                            ;=================================================================
125B'                       NUNTIL:
125B'   55 4E 54 49                 DB      'UNTI','L' OR CLAST
125F'   CC
1260'   124B'                       DW      REPEAT-1
1262'   45                          DB      5 OR IMM
1263'                       UNTIL:
1263'   1108' 128D'                 DW      DOCOMPILER,DOUNTIL
1267'   12D8'                       DW      ASSERT
1269'   01                          DB      1                       ;TEST CHECK VALUE
126A'   1237'                       DW      DORPATCH
126C'   04B6'                       DW      SEMIS
                            ;=================================================================
126E'   02                          DB      2
126F'   FF75                        DW      NELSE-$-1
1271'                       DOELSE:
1271'   1278'                       DW      FJUMP
                            ;=================================================================
1273'   02                          DB      2
1274'   FFCE                        DW      NREPEAT-$-1
1276'                       DOREPEAT:
1276'   1278'                       DW      FJUMP
```

```
                                ;==================================================================
1278'                           FJUMP:
1278'   E1                              POP     HL
1279'   5E                              LD      E,(HL)
127A'   23                              INC     HL
127B'   56                              LD      D,(HL)          ;GET OFFSET
127C'                           OFFSJUMP:
127C'   19                              ADD     HL,DE
127D'   C3 04BA'                        JP      NEXTSUB         ;SET NEW FORTH POINTER
                                ;==================================================================
1280'   02                              DB      2
1281'   FF39                            DW      NIF-$-1
1283'                           DOIF:
1283'   128F'                           DW      IF0JUMP
                                ;==================================================================
1285'   02                              DB      2
1286'   FF46                            DW      NWHILE-$-1
1288'                           DOWHILE:
1288'   128F'                           DW      IF0JUMP
                                ;==================================================================
128A'   02                              DB      2
128B'   FFCF                            DW      NUNTIL-$-1
128D'                           DOUNTIL:
128D'   128F'                           DW      IF0JUMP
                                ;------------------------------------------------------------------
128F'                           IF0JUMP:
128F'   CD 084E'                        CALL    PULLBC
1292'   78                              LD      A,B
1293'   B1                              OR      C               ;TEST FOR 0
1294'                           EQUJUMP:
1294'   28 E2                           JR      Z,FJUMP         ;CONDITION TRUE?
1296'   E1                              POP     HL
1297'   23                              INC     HL
1298'   23                              INC     HL
1299'   C3 04BA'                        JP      NEXTSUB         ;SKIP OFFSET
                                ;==================================================================
129C'   00                              DB      0
129D'   FF74                            DW      NBEGIN-$-1
129F'                           DOBEGIN:
129F'   04B9'                           DW      NEXT
                                ;==================================================================
12A1'   00                              DB      0
12A2'   FF5D                            DW      NTHEN-$-1
12A4'                           DOTHEN:
12A4'   04B9'                           DW      NEXT
                                ;==================================================================
12A6'                           NDO:
12A6'   44 CF                           DB      'D','O' OR CLAST
12A8'   1262'                           DW      UNTIL-1
12AA'   42                              DB      2 OR IMM
12AB'                           DO:
12AB'   1108' 1323'                     DW      DOCOMPILER,DODO
12AF'   0460'                           DW      HERE
12B1'   104B'                           DW      GETBYTE
12B3'   03                              DB      3                       ;SET CHECK VALUE
12B4'   04B6'                           DW      SEMIS
```

```
                                        ;=================================================================
12B6'                                   NLOOP:
12B6'   4C 4F 4F D0                             DB      'LOO','P' OR CLAST
12BA'   12AA'                                   DW      DO-1
12BC'   44                                      DB      4 OR IMM
12BD'                                   LOOP:
12BD'   1108' 1332'                             DW      DOCOMPILER,DOLOOP
12C1'                                   LOOPGOON:
12C1'   12D8'                                   DW      ASSERT
12C3'   03                                      DB      3                       ;TEST CHECK VALUE
12C4'   1237'                                   DW      DORPATCH
12C6'   04B6'                                   DW      SEMIS
                                        ;=================================================================
12C8'                                   NPLUSLOOP:
12C8'   2B 4C 4F 4F                             DB      '+LOO','P' OR CLAST
12CC'   D0
12CD'   12BC'                                   DW      LOOP-1
12CF'   45                                      DB      5 OR IMM
12D0'                                   PLUSLOOP:
12D0'   1108' 133C'                             DW      DOCOMPILER,DOPLUSLOOP
12D4'   1276' FFEA                              DW      DOREPEAT,LOOPGOON-$-1
                                        ;=================================================================
12D8'                                   ASSERT:
12D8'   12DA'                                   DW      $+2

                                                RSTPULL
12DA'   DF                  +                   RST     018H
12DB'   E1                                      POP     HL
12DC'   7E                                      LD      A,(HL)
12DD'   23                                      INC     HL
12DE'   E5                                      PUSH    HL              ;CHECK-VALUE
12DF'   93                                      SUB     E
12E0'   B2                                      OR      D
12E1'   28 4A                                   JR      Z,JNEXT4        ;SAME VALUE ON STACK?
                                                RSTERR  ERRBLK
12E3'   E7                  +                   RST     020H
12E4'   05                  +                   DB      ERRBLK
                                        ;=================================================================
12E5'   C9                                      DB      'I' OR CLAST
12E6'   11AA'                                   DW      DEFINITIONS-1
12E8'   01                                      DB      1
12E9'                                   I:
12E9'   12EB'                                   DW      $+2

12EB'   C1                                      POP     BC
12EC'   D1                                      POP     DE              ;LOOP COUNTER (OR "R")
12ED'   D5                                      PUSH    DE
12EE'   C5                                      PUSH    BC
                                                RSTPUSH
12EF'   D7                  +                   RST     010H
12F0'   FD E9                                   JP      (IY)
                                        ;=================================================================
12F2'   49 A7                                   DB      'I','''' OR CLAST
12F4'   12E8'                                   DW      I-1
12F6'   02                                      DB      2
12F7'                                   ITICK:
```

```
12F7'   12F9'                   DW      $+2

12F9'   21 0004                 LD      HL,4            ;"R2" (SEE "I")
12FC'   18 09                   JR      RGET
                        ;=================================================================
12FE'   CA                      DB      'J' OR CLAST
12FF'   12F6'                   DW      ITICK-1
1301'   01                      DB      1
1302'                   J:
1302'   1304'                   DW      $+2

1304'   21 0006                 LD      HL,6            ;"R3" (SEE "I")
1307'                   RGET:
1307'   39                      ADD     HL,SP
1308'   5E                      LD      E,(HL)
1309'   23                      INC     HL
130A'   56                      LD      D,(HL)          ;GET COUNTER FROM RETURN-STACK
                                RSTPUSH
130B'   D7              +       RST     010H
130C'   FD E9                   JP      (IY)
                        ;=================================================================
130E'   4C 45 41 56             DB      'LEAV','E' OR CLAST
1312'   C5
1313'   1301'                   DW      J-1
1315'   05                      DB      5
1316'                   LEAVE:
1316'   1318'                   DW      $+2

1318'   C1                      POP     BC
1319'   E1                      POP     HL
131A'   E1                      POP     HL
131B'   E5                      PUSH    HL
131C'   E5                      PUSH    HL              ;MAKE COUTERS EQUAL (I = I')
131D'   C5                      PUSH    BC
131E'   FD E9                   JP      (IY)
                        ;=================================================================
1320'   00                      DB      0
1321'   FF84                    DW      NDO-$-1
1323'                   DODO:
1323'   1325'                   DW      $+2

1325'   CD 084E'                CALL    PULLBC
                                RSTPULL
1328'   DF              +       RST     018H
1329'   E1                      POP     HL
132A'   D5                      PUSH    DE
132B'   C5                      PUSH    BC              ;PLACE LIMIT AND INITIAL COUNTER
132C'   E5                      PUSH    HL
132D'                   JNEXT4:
132D'   FD E9                   JP      (IY)
                        ;=================================================================
132F'   02                      DB      2
1330'   FF85                    DW      NLOOP-$-1
1332'                   DOLOOP:
1332'   1334'                   DW      $+2
```

```
1334'   11 0001                         LD      DE,1
1337'   18 06                           JR      LOOPADD
                                ;=================================================================
1339'   02                              DB      2
133A'   FF8D                            DW      NPLUSLOOP-$-1
133C'                           DOPLUSLOOP:
133C'   133E'                           DW      $+2

                                        RSTPULL
133E'   DF                 +            RST     018H
133F'                           LOOPADD:
133F'   C1                              POP     BC
1340'   E1                              POP     HL              ;GET COUNTER
1341'   A7                              AND     A
1342'   ED 5A                           ADC     HL,DE           ;INCREASE (??? DEPENDENT)
1344'   7A                              LD      A,D
1345'   D1                              POP     DE              ;GET FULL VALUE
1346'   37                              SCF
1347'   EA 1358'                        JP      PE,LOOPEND      ;OVERFLOW? (=> END)

134A'   D5                              PUSH    DE
134B'   E5                              PUSH    HL              ;RESTORE VALUES BACK

134C'   07                              RLCA
134D'   30 01                           JR      NC,LOOPCMP
134F'   EB                              EX      DE,HL
1350'                           LOOPCMP:
1350'   CD 0C99'                        CALL    GREATER
1353'   3F                              CCF
1354'   30 02                           JR      NC,LOOPEND      ;NOT FINISHED YET?

1356'   E1                              POP     HL
1357'   E1                              POP     HL              ;REMOVE LOOP VALUES

1358'                           LOOPEND:
1358'   C5                              PUSH    BC
1359'   9F                              SBC     A,A
135A'   C3 1294'                        JP      EQUJUMP
                                ;=================================================================
135D'                           NLBRACKET:
135D'   A8                              DB      '(' OR CLAST
135E'   13D4'                           DW      LSQRBR-1
1360'   41                              DB      1 OR IMM
1361'                           LBRACKET:
1361'   1108' 1379'                     DW      DOCOMPILER,DOLBRACKET
1365'   104B'                           DW      GETBYTE
1367'   29                              DB      ')'
1368'                           LBREND:
1368'   0460' 0885'                     DW      HERE,SWAP,ALLOT2,SAVETEXT
136C'   0F83' 139F'
1370'   0885' 08C1'                     DW      SWAP,EXCLAM             ;SAVE AFTER-TEXT ADDRESS
1374'   04B6'                           DW      SEMIS
                                ;=================================================================
1376'   FF                              DB      -1
1377'   FFE5                            DW      NLBRACKET-$-1
1379'                           DOLBRACKET:
```

```
1379'   137B'                   DW      $+2

137B'   E1                      POP     HL
137C'   5E                      LD      E,(HL)
137D'   23                      INC     HL
137E'   56                      LD      D,(HL)          ;GET OFFSET
137F'   13                      INC     DE
1380'   C3 127C'                JP      OFFSJUMP
                        ;===================================================================
1383'                   NPTSTR:
1383'   2E A2                   DB      '.','"' OR CLAST
1385'   1360'                   DW      LBRACKET-1
1387'   42                      DB      2 OR IMM
1388'                   PTSTR:
1388'   1108' 1396'             DW      DOCOMPILER,DOPTSTR
138C'   104B'                   DW      GETBYTE
138E'   22                      DB      '"'
138F'   1276' FFD6              DW      DOREPEAT,LBREND-$-1
                        ;===================================================================
1393'   FF                      DB      -1
1394'   FFEE                    DW      NPTSTR-$-1
1396'                   DOPTSTR:
1396'   1398'                   DW      $+2

1398'   D1                      POP     DE
1399'   CD 0979'                CALL    TYPEDE          ;TYPE STRING
139C'   D5                      PUSH    DE
139D'   FD E9                   JP      (IY)
                        ;===================================================================
139F'                   SAVETEXT:
139F'   13A1'                   DW      $+2

13A1'                   STLOOP:
                                RSTPULL
13A1'   DF              +       RST     018H
13A2'   D5                      PUSH    DE
13A3'   CD 05E1'                CALL    CWORD           ;END SEARCH
13A6'   62                      LD      H,D
13A7'   6B                      LD      L,E
13A8'   09                      ADD     HL,BC
13A9'   7E                      LD      A,(HL)
13AA'   E1                      POP     HL
13AB'   BD                      CP      L
13AC'   28 0A                   JR      Z,STFND         ;END FOUND?

13AE'   EB                      EX      DE,HL
                                RSTPUSH
13AF'   D7              +       RST     010H
13B0'   11 0578'                LD      DE,RETYPE
13B3'   CD 1815'                CALL    EXECDE
13B6'   18 E9                   JR      STLOOP          ;TRY AGAIN

13B8'                   STFND:
13B8'   D5                      PUSH    DE
13B9'   C5                      PUSH    BC
13BA'   2A 3C37                 LD      HL,(STKBOT)     ;BORDER WITH SAFE-GAP
```

```
13BD'   CD 0F9E'                        CALL    ALLOC           ;RESERVE MEMORY
13C0'   C1                              POP     BC
13C1'   D1                              POP     DE
13C2'   D5                              PUSH    DE
13C3'   C5                              PUSH    BC
13C4'   EB                              EX      DE,HL
13C5'   ED B0                           LDIR                    ;MOVE TEXT
13C7'   C1                              POP     BC
13C8'   50                              LD      D,B
13C9'   59                              LD      E,C
                                        RSTPUSH
13CA'   D7              +               RST     010H
13CB'   D1                              POP     DE
13CC'   CD 07DA'                        CALL    BLWORD          ;CLEAR INPUT
13CF'   FD E9                           JP      (IY)
                                ;=================================================================
13D1'   DB                              DB      '[' OR CLAST
13D2'   12CF'                           DW      PLUSLOOP-1
13D4'   41                              DB      1 OR IMM
13D5'                           LSQRBR:
13D5'   13D7'                           DW      $+2

13D7'   DD CB 3E B6                     RES     6,(IX+FLAGS-MEMBEG)     ;TURN OFF COMPILATION
13DB'   FD E9                           JP      (IY)
                                ;=================================================================
13DD'   DD                              DB      ']' OR CLAST
13DE'   1315'                           DW      LEAVE-1
13E0'   01                              DB      1
13E1'                           RSQRBR:
13E1'   13E3'                           DW      $+2

13E3'   DD CB 3E F6                     SET     6,(IX+FLAGS-MEMBEG)     ;TURN ON COMPILATION
13E7'   FD E9                           JP      (IY)
                                ;=================================================================
13E9'   45 58 49 D4                     DB      'EXI','T' OR CLAST
13ED'   1387'                           DW      PTSTR-1
13EF'   04                              DB      4
13F0'                           EXIT:
13F0'   04B8'                           DW      RSEMIS
                                ;=================================================================
0000                            RDONAME EQU     0       ;POINTER TO OLD WORD NAME
0002                            RDOCODE EQU     2       ;POINTER TO OLD WORD CODE FIELD
0004                            RDNCODE EQU     4       ;POINTER TO NEW WORD CODE FIELD
0004                            RDDNAME EQU     4       ; DIFFERENCE OF NAME LENGTHS
0006                            RDNRUN  EQU     6       ;NEW WORD 0 / RUN ADDRESS
0008                            RDOEND  EQU     8       ;OLD WORD END POINTER
000A                            RDNEND  EQU     10      ;NEW WORD END POINTER
000A                            RDDLEN  EQU     10      ; LENGTH DIFFERENCE
000C                            RDNNAME EQU     12      ;NEW WORD POINTER TO NAME
                                ;=================================================================
13F2'   52 45 44 45                     DB      'REDEFIN','E' OR CLAST
13F6'   46 49 4E C5
13FA'   13EF'                           DW      EXIT-1
13FC'   08                              DB      8
13FD'                           REDEFINE:
13FD'   13FF'                           DW      $+2
```

```
13FF'   CD 0F2E'                    CALL    LINKHERE
1402'   2A 3C31                     LD      HL,(VCURRENT)
1405'   5E                          LD      E,(HL)
1406'   23                          INC     HL
1407'   56                          LD      D,(HL)
1408'   EB                          EX      DE,HL
1409'   23                          INC     HL
140A'   22 2705                     LD      (PADMEM+RDNCODE),HL     ;NEW WORD CODEFIELD

140D'   E5                          PUSH    HL
140E'   CD 15C0'                    CALL    PTR2ADDR
1411'   22 270D                     LD      (PADMEM+RDNNAME),HL
1414'   ED 43 2707                  LD      (PADMEM+RDNRUN),BC
1418'   ED 53 270B                  LD      (PADMEM+RDNEND),DE      ;GET ADDRESS

141C'   2A 3C37                     LD      HL,(STKBOT)
141F'   ED 52                       SBC     HL,DE
1421'   C2 14DA'                    JP      NZ,DICTERR      ;WORD NOT THE LATEST?

1424'   D1                          POP     DE
                                    RSTPUSH                 ;WORD TO BE REDEFINED
1425'   D7              +           RST     010H
1426'   CD 04B9'                    CALL    NEXT
1429'   1610' 063D'                 DW      RESCURR,FIND,SEMICODE
142D'   1A0E'
                                    RSTPULL                 ;OLD WORD CODEFIELD ADR.
142F'   DF              +           RST     018H
1430'   21 C3AF                     LD      HL,-FREEMEM
1433'   19                          ADD     HL,DE
1434'   D2 14CF'                    JP      NC,REDEFABORT   ;WORD NOT IN RAM?

1437'   EB                          EX      DE,HL
1438'   22 2703                     LD      (PADMEM+RDOCODE),HL
143B'   CD 15C0'                    CALL    PTR2ADDR        ;GET ADDRESS
143E'   22 2701                     LD      (PADMEM+RDONAME),HL
1441'   E5                          PUSH    HL              ;(SEE BELLOW !!!)
1442'   ED 53 2709                  LD      (PADMEM+RDOEND),DE
1446'   78                          LD      A,B
1447'   B1                          OR      C
1448'   ED 5B 2707                  LD      DE,(PADMEM+RDNRUN)
144C'   28 04                       JR      Z,RDGOON1       ;OLD WITH NO SPECIAL RUN PART?
144E'   7A                          LD      A,D
144F'   B3                          OR      E
1450'   28 7D                       JR      Z,REDEFABORT    ;NEW WITH NO SPECIAL RUN PART?

1452'                       RDGOON1:
1452'   E1                          POP     HL
1453'   ED 4B 270D                  LD      BC,(PADMEM+RDNNAME)
1457'   ED 42                       SBC     HL,BC
1459'   EB                          EX      DE,HL
145A'   19                          ADD     HL,DE
145B'   22 2707                     LD      (PADMEM+RDNRUN),HL      ;UPDATE RUN ADDRESS

145E'   2A 270B                     LD      HL,(PADMEM+RDNEND)
1461'   19                          ADD     HL,DE
```

```
1462'   ED 4B 2709              LD      BC,(PADMEM+RDOEND)
1466'   A7                      AND     A
1467'   ED 42                   SBC     HL,BC
1469'   22 270B                 LD      (PADMEM+RDDLEN),HL      ;CALCULATE LENGTH DIFFERENCE

146C'   01 002E                 LD      BC,46
146F'   09                      ADD     HL,BC
1470'   CB 7C                   BIT     7,H
1472'   20 0B                   JR      NZ,RDGOON2      ;AT LEAST 47 BYTE SMALLER?

1474'   ED 4B 3C3B              LD      BC,(SPARE)
1478'   09                      ADD     HL,BC
1479'   38 54                   JR      C,REDEFABORT
147B'   ED 72                   SBC     HL,SP
147D'   30 50                   JR      NC,REDEFABORT   ;INSUFFICIENT MEMORY?

147F'                   RDGOON2:
147F'   2A 2703                 LD      HL,(PADMEM+RDOCODE)
1482'   E5                      PUSH    HL
1483'   2B                      DEC     HL
1484'   2B                      DEC     HL
1485'   46                      LD      B,(HL)
1486'   2B                      DEC     HL
1487'   4E                      LD      C,(HL)
1488'   2A 2705                 LD      HL,(PADMEM+RDNCODE)
148B'   E5                      PUSH    HL
148C'   2B                      DEC     HL
148D'   2B                      DEC     HL
148E'   70                      LD      (HL),B
148F'   2B                      DEC     HL
1490'   71                      LD      (HL),C          ;LINK WORDS
1491'   E1                      POP     HL
1492'   19                      ADD     HL,DE
1493'   C1                      POP     BC
1494'   A7                      AND     A
1495'   ED 42                   SBC     HL,BC
1497'   22 2705                 LD      (PADMEM+RDDNAME),HL     ;CALC NAME LENGTH DIFF.

149A'   ED 5B 2701              LD      DE,(PADMEM+RDONAME)
149E'   2A 2709                 LD      HL,(PADMEM+RDOEND)
14A1'   A7                      AND     A
14A2'   ED 52                   SBC     HL,DE
14A4'   44                      LD      B,H
14A5'   4D                      LD      C,L
14A6'   D5                      PUSH    DE
14A7'   C5                      PUSH    BC
14A8'   CD 14DC'                CALL    DELWORD         ;REMOVE OLD WORD

14AB'   2A 270B                 LD      HL,(PADMEM+RDDLEN)
14AE'   C1                      POP     BC
14AF'   09                      ADD     HL,BC
14B0'   44                      LD      B,H
14B1'   4D                      LD      C,L
14B2'   E1                      POP     HL
14B3'   C5                      PUSH    BC
14B4'   CD 0F9E'                CALL    ALLOC           ;RESERVE MEMORY FOR NEW WORD
```

```
14B7'   EB                          EX      DE,HL
14B8'   2A 270D                     LD      HL,(PADMEM+RDNNAME)
14BB'   ED 4B 270B                  LD      BC,(PADMEM+RDDLEN)
14BF'   09                          ADD     HL,BC           ;ADJUST START-ADDRESS
14C0'   C1                          POP     BC
14C1'   C5                          PUSH    BC
14C2'   E5                          PUSH    HL
14C3'   ED B0                       LDIR                    ;COPY NEW WORD
14C5'   D1                          POP     DE
14C6'   C1                          POP     BC
14C7'   CD 14DC'                    CALL    DELWORD         ;REMOVE ORIGINAL WORD

14CA'   CD 14F8'                    CALL    CORRCURR        ;CORRECT POINTER
14CD'   FD E9                       JP      (IY)

14CF'                       REDEFABORT:
14CF'   2A 3C31                     LD      HL,(VCURRENT)
14D2'   ED 5B 2705                  LD      DE,(PADMEM+RDNCODE)
14D6'   1B                          DEC     DE
14D7'   73                          LD      (HL),E
14D8'   23                          INC     HL
14D9'   72                          LD      (HL),D          ;SET CURRENT DICTIONARY
14DA'                       DICTERR:
                                    RSTERR  ERRDICT
14DA'   E7              +           RST     020H
14DB'   0B              +           DB      ERRDICT
                            ;=================================================================
14DC'                       DELWORD:
14DC'   2A 3C37                     LD      HL,(STKBOT)
14DF'   A7                          AND     A
14E0'   ED 42                       SBC     HL,BC
14E2'   22 3C37                     LD      (STKBOT),HL     ;LOWERED HERE

14E5'   2A 3C3B                     LD      HL,(SPARE)
14E8'   ED 42                       SBC     HL,BC
14EA'   22 3C3B                     LD      (SPARE),HL      ;LOWERED SPARE

14ED'   ED 52                       SBC     HL,DE
14EF'   C8                          RET     Z               ;WAS IT LASTEST WORD?

14F0'   C5                          PUSH    BC
14F1'   44                          LD      B,H
14F2'   4D                          LD      C,L
14F3'   E1                          POP     HL
14F4'   19                          ADD     HL,DE
14F5'   ED B0                       LDIR                    ;MOVE WHAT FOLLOWS
14F7'   C9                          RET
                            ;-----------------------------------------------------------------
14F8'                       CORRCURR:
14F8'   01 3C31                     LD      BC,VCURRENT
14FB'   CD 1557'                    CALL    CORRPTR
14FE'   CD 1557'                    CALL    CORRPTR         ;AJUST PONTER IN CURRENT

1501'   01 3C40                     LD      BC,DICT1ST
1504'                       CORRDICT:
1504'   2A 3C37                     LD      HL,(STKBOT)
```

```
1507'   37                              SCF
1508'   ED 42                           SBC     HL,BC
150A'   D8                              RET     C               ;REACHED THE END?

150B'                           CDLOOP:
150B'   0A                              LD      A,(BC)
150C'   17                              RLA
150D'   03                              INC     BC
150E'   30 FB                           JR      NC,CDLOOP       ;SKIP NAME

1510'   03                              INC     BC
1511'   03                              INC     BC
1512'   CD 1557'                        CALL    CORRPTR         ;CORRECT END ADDRESS
1515'   03                              INC     BC
1516'   CD 1557'                        CALL    CORRPTR         ;FIRST WORD ON DICT.
1519'   CD 15FB'                        CALL    JUMPDE
151C'   0EC3'                           DW      DOCOL
151E'   1C                              DB      CDCOLON-$
151F'   1085'                           DW      DODEFINER
1521'   16                              DB      CDDEFCOM-$
1522'   1108'                           DW      DOCOMPILER
1524'   13                              DB      CDDEFCOM-$
1525'   11B5'                           DW      SETCONTEXT
1527'   18                              DB      CDSETCTXT-$
1528'   0000                            DW      0

152A'   21 FFF9                         LD      HL,-7
152D'   09                              ADD     HL,BC
152E'   4E                              LD      C,(HL)
152F'   23                              INC     HL
1530'   46                              LD      B,(HL)
1531'   2B                              DEC     HL
1532'   09                              ADD     HL,BC           ;LINK TO PREVIOUS DICT.
1533'   44                              LD      B,H
1534'   4D                              LD      C,L
1535'   18 CD                           JR      CORRDICT

1537'                           CDDEFCOM:
1537'   CD 1557'                        CALL    CORRPTR
153A'                           CDCOLON:
153A'   CD 1548'                        CALL    CORRWORD
153D'   18 C5                           JR      CORRDICT

153F'                           CDSETCTXT:
153F'   CD 1557'                        CALL    CORRPTR
1542'   03                              INC     BC
1543'   CD 1557'                        CALL    CORRPTR
1546'   18 BC                           JR      CORRDICT
                                ;-----------------------------------------------------------------
1548'                           CORRWORD:
1548'   CD 1557'                        CALL    CORRPTR
154B'   21 04B6'                        LD      HL,SEMIS
154E'   A7                              AND     A
154F'   ED 52                           SBC     HL,DE
1551'   C8                              RET     Z               ;END OF WORD FOUND?
```

```
1552'   CD 159E'                        CALL    SKIPOFFS
1555'   18 F1                           JR      CORRWORD
                                ;-----------------------------------------------------------------
1557'                           CORRPTR:
1557'   0A                              LD      A,(BC)
1558'   5F                              LD      E,A
1559'   03                              INC     BC
155A'   0A                              LD      A,(BC)
155B'   57                              LD      D,A
155C'   0B                              DEC     BC              ;GET ADDRESS

155D'   CD 1568'                        CALL    CORRADDR

1560'   EB                              EX      DE,HL
1561'   7B                              LD      A,E
1562'   02                              LD      (BC),A
1563'   03                              INC     BC
1564'   7A                              LD      A,D
1565'   02                              LD      (BC),A          ;SAVE CORRECTED POINTER
1566'   03                              INC     BC
1567'   C9                              RET
                                ;-----------------------------------------------------------------
1568'                           CORRADDR:
1568'   2A 2701                         LD      HL,(PADMEM+RDONAME)
156B'   A7                              AND     A
156C'   ED 52                           SBC     HL,DE
156E'   62                              LD      H,D
156F'   6B                              LD      L,E
1570'   D0                              RET     NC              ;OLD WORD => NO ADJUSTMENT

1571'   2A 2709                         LD      HL,(PADMEM+RDOEND)
1574'   ED 52                           SBC     HL,DE
1576'   30 0C                           JR      NC,CAWORD       ;REDEFINED WORD?

1578'   2A 270D                         LD      HL,(PADMEM+RDNNAME)
157B'   ED 52                           SBC     HL,DE
157D'   38 13                           JR      C,CADICT        ;OTHER DICTIONARY?

157F'   2A 270B                         LD      HL,(PADMEM+RDDLEN)
1582'   19                              ADD     HL,DE
1583'   C9                              RET                     ;NEW => ADJUST BY DIFFERENCE

1584'                           CAWORD:
1584'   2A 2703                         LD      HL,(PADMEM+RDOCODE)
1587'   ED 52                           SBC     HL,DE
1589'   2A 2707                         LD      HL,(PADMEM+RDNRUN)
158C'   D8                              RET     C               ;HAS RUN PART => NEW ADDRESS

158D'   2A 2705                         LD      HL,(PADMEM+RDDNAME)
1590'   19                              ADD     HL,DE
1591'   C9                              RET                     ;ADJUST TO NAME DIFFERENCE

1592'                           CADICT:
1592'   2A 2701                         LD      HL,(PADMEM+RDONAME)
1595'   19                              ADD     HL,DE
1596'   ED 5B 270D                      LD      DE,(PADMEM+RDNNAME)
```

```
159A'   A7                          AND     A
159B'   ED 52                       SBC     HL,DE
159D'   C9                          RET                     ;ADJUST TO LENGTH DIFFERENCE
                            ;-----------------------------------------------------------------
159E'                       SKIPOFFS:
159E'   1B                          DEC     DE
159F'   1A                          LD      A,(DE)
15A0'   17                          RLA
15A1'   D0                          RET     NC              ;NORMAL FORTH-WORD ?

15A2'                       SKOFFS2:
15A2'   1B                          DEC     DE
15A3'   1B                          DEC     DE
15A4'   1A                          LD      A,(DE)          ;GET OFFSET
15A5'   6F                          LD      L,A
15A6'   26 00                       LD      H,0
15A8'   3C                          INC     A
15A9'   20 06                       JR      NZ,SKOGOON      ;OFFSET-BYTE VALID ?
15AB'   0A                          LD      A,(BC)
15AC'   6F                          LD      L,A
15AD'   03                          INC     BC
15AE'   0A                          LD      A,(BC)
15AF'   67                          LD      H,A
15B0'   03                          INC     BC              ;GET OFFSET IN CODE

15B1'                       SKOGOON:
15B1'   09                          ADD     HL,BC
15B2'   44                          LD      B,H
15B3'   4D                          LD      C,L             ;SAVE THE NEW ADDRESS
15B4'   C9                          RET
                            ;-----------------------------------------------------------------
15B5'                       NFA:
15B5'   15B7'                       DW      $+2
                                    RSTPULL
15B7'   DF              +           RST     018H
15B8'   EB                          EX      DE,HL
15B9'   CD 15E7'                    CALL    FPTR2NAME
15BC'   EB                          EX      DE,HL
                                    RSTPUSH
15BD'   D7              +           RST     010H
15BE'   FD E9                       JP      (IY)
                            ;-----------------------------------------------------------------
15C0'                       PTR2ADDR:
15C0'   E5                          PUSH    HL
15C1'   5E                          LD      E,(HL)
15C2'   23                          INC     HL
15C3'   56                          LD      D,(HL)          ;GET THE FIRST WORD ADDRESS
15C4'   CD 15FB'                    CALL    JUMPDE
15C7'   1108'                       DW      DOCOMPILER
15C9'   0B                          DB      P2ARUN-$
15CA'   1085'                       DW      DODEFINER
15CC'   08                          DB      P2ARUN-$
15CD'   0000                        DW      0

15CF'   01 0000                     LD      BC,0            ;NO SPECIAL RUN PART
15D2'   18 07                       JR      P2AGOON
```

```
15D4'                     P2ARUN:
15D4'   E1                        POP     HL
15D5'   E5                        PUSH    HL
15D6'   23                        INC     HL
15D7'   23                        INC     HL
15D8'   4E                        LD      C,(HL)
15D9'   23                        INC     HL
15DA'   46                        LD      B,(HL)          ;GET RUNTIME ADDRESS

15DB'                     P2AGOON:
15DB'   E1                        POP     HL
15DC'   E5                        PUSH    HL
15DD'   2B                        DEC     HL
15DE'   2B                        DEC     HL
15DF'   2B                        DEC     HL
15E0'   2B                        DEC     HL
15E1'   56                        LD      D,(HL)
15E2'   2B                        DEC     HL
15E3'   5E                        LD      E,(HL)
15E4'   19                        ADD     HL,DE
15E5'   EB                        EX      DE,HL           ;CALCULATE POINTER BEHIND WORD
15E6'   E1                        POP     HL
                          ;-----------------------------------------------------------------
15E7'                     FPTR2NAME:
15E7'   2B                        DEC     HL
15E8'                     PTR2NAME:
15E8'   7C                        LD      A,H
15E9'   FE 3C                     CP      MEMBEG SHR 8
15EB'   7E                        LD      A,(HL)
15EC'   CB B7                     RES     6,A             ;CLEAR IMMEDIATE BIT
15EE'   38 02                     JR      C,P2NGOON
15F0'   C6 02                     ADD     A,2             ;MORE FOR WORDS IN RAM
15F2'                     P2NGOON:
15F2'   2B                        DEC     HL
15F3'   2B                        DEC     HL              ;SKIP OVER LINK
15F4'                     P2NLOOP:
15F4'   2B                        DEC     HL
15F5'   3D                        DEC     A
15F6'   20 FC                     JR      NZ,P2NLOOP      ;POINT TO START OF NAME
15F8'   C9                        RET
                          ;=================================================================
15F9'                     JDELOOP:
15F9'   23                        INC     HL              ;SKIP OFFSET
15FA'   E5                        PUSH    HL
15FB'                     JUMPDE:
15FB'   E1                        POP     HL
15FC'   7E                        LD      A,(HL)
15FD'   23                        INC     HL
15FE'   E5                        PUSH    HL
15FF'   66                        LD      H,(HL)
1600'   6F                        LD      L,A             ;GET NEXT POINTER

1601'   B4                        OR      H
1602'   C8                        RET     Z               ;0 ? (HRM-HRM, FOR A "NOP" !!!)
```

```
1603'   ED 52                       SBC     HL,DE
1605'   E1                          POP     HL
1606'   23                          INC     HL
1607'   20 F0                       JR      NZ,JDELOOP      ;POINTER NOT YET REACHED?

1609'   D5                          PUSH    DE
160A'   16 00                       LD      D,0
160C'   5E                          LD      E,(HL)          ;GET OFFSET
160D'   19                          ADD     HL,DE
160E'   D1                          POP     DE
160F'   E9                          JP      (HL)            ;JUMP TO CODE
                            ;=================================================================
1610'                       RESCURR:
1610'   0EC3'                       DW      DOCOL
1612'   0E1F' 0E29'                 DW      ONEMINUS,TWOMINUS,AT
1616'   08B3'
1618'   0480' 08B3'                 DW      CURRENT,AT,EXCLAM       ;SET IT AS CURRENT
161C'   08C1'
161E'   04B6'                       DW      SEMIS
                            ;=================================================================
1620'                       FINDWORD:
1620'   CD 04B9'                    CALL    NEXT
1623'   063D'                       DW      FIND
1625'   1A0E'                       DW      SEMICODE
                                    RSTPULL                 ;CODE FIELD ADDRESS
1627'   DF              +           RST     018H
1628'   21 C3AF                     LD      HL,-FREEMEM
162B'   19                          ADD     HL,DE
162C'   D8                          RET     C               ;WORD FOUND?
                                    RSTERR  ERRFIND
162D'   E7              +           RST     020H
162E'   0D              +           DB      ERRFIND
                            ;=================================================================
162F'   46 4F 52 47                 DB      'FORGE','T' OR CLAST
1633'   45 D4
1635'   13FC'                       DW      REDEFINE-1
1637'   06                          DB      6
1638'                       FORGET:
1638'   163A'                       DW      $+2

163A'   2A 3C31                     LD      HL,(VCURRENT)
163D'   ED 5B 3C33                  LD      DE,(VCONTEXT)
1641'   A7                          AND     A
1642'   ED 52                       SBC     HL,DE
1644'   C2 14DA'                    JP      NZ,DICTERR      ;DIFFERENT DICTIONARIES?

1647'   CD 1620'                    CALL    FINDWORD
164A'   21 FFFB                     LD      HL,-5
164D'   19                          ADD     HL,DE
164E'   22 3C39                     LD      (DICT),HL
1651'   DD CB 3E D6                 SET     2,(IX+FLAGS-MEMBEG)     ;TURN COMPILE MODE ON
                                    RSTERR  ERRNONE
1655'   E7              +           RST     020H
1656'   FF              +           DB      ERRNONE
                            ;=================================================================
1657'   45 44 49 D4                 DB      'EDI','T' OR CLAST
```

```
165B'   1637'                       DW      FORGET-1
165D'   04                          DB      4
165E'                       EDIT:
165E'   1660'                       DW      $+2

1660'   CD 1620'                    CALL    FINDWORD
1663'   DD CB 3E DE                 SET     3,(IX+FLAGS-MEMBEG)     ;SET EDIT MODE
1667'   18 0C                       JR      EDITLIST
                            ;=================================================================
1669'   4C 49 53 D4                 DB      'LIS','T' OR CLAST
166D'   165D'                       DW      EDIT-1
166F'   04                          DB      4
1670'                       LIST:
1670'   1672'                       DW      $+2

1672'   CD 1620'                    CALL    FINDWORD
                            ;-----------------------------------------------------------------
1675'                       EDITLIST:
1675'   3E 0D                       LD      A,CCR
                                    RSTEMIT
1677'   CF              +           RST     008H

1678'   DD CB 3E 5E                 BIT     3,(IX+FLAGS-MEMBEG)
167C'   D5                          PUSH    DE
167D'   C4 02D8'                    CALL    NZ,DCCLEAR      ;EDIT MODE ?
1680'   C1                          POP     BC

1681'   0A                          LD      A,(BC)
1682'   5F                          LD      E,A
1683'   03                          INC     BC
1684'   0A                          LD      A,(BC)
1685'   57                          LD      D,A
1686'   0B                          DEC     BC
1687'   CD 15FB'                    CALL    JUMPDE
168A'   0EC3'                       DW      DOCOL
168C'   0B                          DB      ELCOLON-$
168D'   1108'                       DW      DOCOMPILER
168F'   0D                          DB      ELCOMPILER-$
1690'   1085'                       DW      DODEFINER
1692'   1F                          DB      ELDEFINER-$
1693'   0000                        DW      0
                                    RSTERR  ERRLIST
1695'   E7              +           RST     020H
1696'   0E              +           DB      ERRLIST
                            ;-----------------------------------------------------------------
1697'                       ELCOLON:
1697'   21 0002                     LD      HL,2
169A'   18 18                       JR      ELOUT
                            ;-----------------------------------------------------------------
169C'                       ELCOMPILER:
169C'   D5                          PUSH    DE

169D'   21 0002                     LD      HL,2
16A0'   09                          ADD     HL,BC
16A1'   7E                          LD      A,(HL)
16A2'   23                          INC     HL
```

```
16A3'   66                              LD      H,(HL)
16A4'   6F                              LD      L,A             ;DOCOMPILER BEHIND ADDRESS

16A5'   2B                              DEC     HL
16A6'   2B                              DEC     HL
16A7'   2B                              DEC     HL
16A8'   6E                              LD      L,(HL)
16A9'   7D                              LD      A,L
16AA'   07                              RLCA
16AB'   9F                              SBC     A,A
16AC'   67                              LD      H,A             ;CODEBYTE ON 16 BIT
16AD'   CD 180E'                        CALL    PNTHL

16B0'   D1                              POP     DE
                                ;-----------------------------------------------------------------
16B1'                           ELDEFINER:
16B1'   21 0004                         LD      HL,4
                                ;-----------------------------------------------------------------
16B4'                           ELOUT:
16B4'   09                              ADD     HL,BC
16B5'   E5                              PUSH    HL
16B6'   C5                              PUSH    BC
16B7'   CD 17E4'                        CALL    OUTWORD         ;PRINT ":" AND MORE
16BA'   D1                              POP     DE
16BB'   C1                              POP     BC
16BC'   CD 17E4'                        CALL    OUTWORD         ;TYPE NAME

16BF'   DD 36 14 01                     LD      (IX+LPIBUF-MEMBEG),1    ;INSERT 1 CHAR
16C3'                           ELMLOOP:
16C3'   DD 36 16 10                     LD      (IX+LPLCNT-MEMBEG),16   ;16 LINES
16C7'                           ELLLOOP:
16C7'   CD 1708'                        CALL    LISTPGM
16CA'   38 06                           JR      C,ELREADY       ;FULL WORD LISTED?
16CC'   DD 35 16                        DEC     (IX+LPLCNT-MEMBEG)
16CF'   F2 16C7'                        JP      P,ELLLOOP       ;NOT ALL LINES USED YET?

16D2'                           ELREADY:
16D2'   DD CB 3E 5E                     BIT     3,(IX+FLAGS-MEMBEG)
16D6'   20 10                           JR      NZ,ELEDIT       ;EDIT MODE ?

16D8'   38 28                           JR      C,ELQUIT        ;WORT FERTIG GELISTET ?

16DA'   21 3C26                         LD      HL,KEYCOD
16DD'   36 00                           LD      (HL),0
16DF'                           ELACK:
16DF'   7E                              LD      A,(HL)
16E0'   A7                              AND     A
16E1'   28 FC                           JR      Z,ELACK         ;WAIT FOR CONFIRMATION

16E3'   CD 04E4'                        CALL    USERBREAK
16E6'   18 DB                           JR      ELMLOOP         ;KEEP IT GOING

16E8'                           ELEDIT:
16E8'   F5                              PUSH    AF
16E9'   DD CB 3E 9E                     RES     3,(IX+FLAGS-MEMBEG)     ;QUICK NO "EDIT"
16ED'   C5                              PUSH    BC
```

```
16EE'   CD 04B9'                        CALL    NEXT
16F1'   0578' 0506'                     DW      RETYPE,LINE
16F5'   1A0E'                           DW      SEMICODE        ;EDIT

16F7'   DD CB 3E DE                     SET     3,(IX+FLAGS-MEMBEG)     ;SET EDIT MODE
16FB'   CD 02D8'                        CALL    DCCLEAR
16FE'   C1                              POP     BC
16FF'   F1                              POP     AF
1700'   30 C1                           JR      NC,ELMLOOP      ;WORD LIST NOT FINISHED?
1702'                           ELQUIT:
1702'   DD CB 3E 9E                     RES     3,(IX+FLAGS-MEMBEG)     ;SET EDIT MODE OFF
1706'   FD E9                           JP      (IY)
                                ;-------------------------------------------------------------
1708'                           LISTPGM:
1708'   3A 3C14                         LD      A,(LPIBUF)
170B'   32 3C15                         LD      (LPIACT),A      ;GET INDENTATION

170E'   DD 36 13 05                     LD      (IX+LPICNT-MEMBEG),5    ;TO 5 WORDS GROUPS
1712'                           LPLOOP:
1712'   0A                              LD      A,(BC)
1713'   5F                              LD      E,A
1714'   03                              INC     BC
1715'   0A                              LD      A,(BC)
1716'   57                              LD      D,A
1717'   03                              INC     BC
1718'   CD 15FB'                        CALL    JUMPDE          ;GET NEXT WORD
171B'   1283'                           DW      DOIF
171D'   40                              DB      LPIINC-$
171E'   1271'                           DW      DOELSE
1720'   44                              DB      LPILEFT-$
1721'   12A4'                           DW      DOTHEN
1723'   48                              DB      LPIDEC-$
1724'   129F'                           DW      DOBEGIN
1726'   37                              DB      LPIINC-$
1727'   128D'                           DW      DOUNTIL
1729'   42                              DB      LPIDEC-$
172A'   1288'                           DW      DOWHILE
172C'   38                              DB      LPILEFT-$
172D'   1276'                           DW      DOREPEAT
172F'   3C                              DB      LPIDEC-$
1730'   1323'                           DW      DODO
1732'   2B                              DB      LPIINC-$
1733'   1332'                           DW      DOLOOP
1735'   36                              DB      LPIDEC-$
1736'   133C'                           DW      DOPLUSLOOP
1738'   33                              DB      LPIDEC-$
1739'   10E8'                           DW      DODOESGT
173B'   29                              DB      LPILEFT-$
173C'   1140'                           DW      DORUNSGT
173E'   26                              DB      LPILEFT-$
173F'   1011'                           DW      GETWORD
1741'   3B                              DB      LPWORD-$
1742'   1064'                           DW      GETFLOAT
1744'   47                              DB      LPFLOAT-$
1745'   104B'                           DW      GETBYTE
```

```
1747'   51                      DB      LPBYTE-$
1748'   1379'                   DW      DOLBRACKET
174A'   62                      DB      LPLBRACKET-$
174B'   1396'                   DW      DOPTSTR
174D'   63                      DB      LPPTSTR-$
174E'   04B6'                   DW      SEMIS
1750'   54                      DB      LPSEMIS-$
1751'   0000                    DW      0

1753'                   LPOUT:
1753'   CD 17E1'                CALL    OUTWORDI
1756'                   LPNEXT:
1756'   DD 35 13                DEC     (IX+LPICNT-MEMBEG)
1759'   20 B7                   JR      NZ,LPLOOP       ;LIMIT NUMBER OF WORDS

175B'   A7                      AND     A               ;WORD NOT FULLY LISTED
175C'   C9                      RET

175D'                   LPIINC:
175D'   2A 3C14                 LD      HL,(LPIBUF)
1760'   65                      LD      H,L
1761'   2C                      INC     L               ;INCREASE INDENT
1762'   18 0C                   JR      LPINDENT

1764'                   LPILEFT:
1764'   2A 3C14                 LD      HL,(LPIBUF)
1767'   65                      LD      H,L
1768'   25                      DEC     H               ;USE PREVIOUS INDENT
1769'   18 05                   JR      LPINDENT

176B'                   LPIDEC:
176B'   2A 3C14                 LD      HL,(LPIBUF)
176E'   2D                      DEC     L
176F'   65                      LD      H,L             ;DECREASE INDENT
1770'                   LPINDENT:
1770'   22 3C14                 LD      (LPIBUF),HL

1773'   DD 36 13 01             LD      (IX+LPICNT-MEMBEG),1    ;ONLY THIS WORD

1777'   DD 35 16                DEC     (IX+LPLCNT-MEMBEG)      ;DONE FOR NOW
177A'   18 D7                   JR      LPOUT

177C'                   LPWORD:
177C'   CD 17DA'                CALL    LPNXTWRD
                                RSTPUSH
177F'   D7              +       RST     010H
1780'   11 09B3'                LD      DE,PNT
1783'                   LPNUMBER:
1783'   CD 17C1'                CALL    OUTINDENT
1786'   CD 1815'                CALL    EXECDE          ;OUTPUT VALUE
1789'   18 CB                   JR      LPNEXT

178B'                   LPFLOAT:
178B'   CD 17DA'                CALL    LPNXTWRD
                                RSTPUSH
178E'   D7              +       RST     010H
```

```
178F'   CD 17DA'                    CALL    LPNXTWRD
                                    RSTPUSH
1792'   D7              +           RST     010H
1793'   11 0AAF'                    LD      DE,FPNT
1796'   18 EB                       JR      LPNUMBER

1798'                       LPBYTE:
1798'   0A                          LD      A,(BC)
1799'   F5                          PUSH    AF
179A'   CD 17E1'                    CALL    OUTWORDI
179D'   F1                          POP     AF
                                    RSTEMIT
179E'   CF              +           RST     008H
179F'   3E 20                       LD      A,' '
                                    RSTEMIT
17A1'   CF              +           RST     008H
17A2'   18 B2                       JR      LPNEXT

17A4'                       LPSEMIS:
17A4'   CD 1808'                    CALL    ROMTXT
17A7'   0D 3B 8D                    DB      CCR,';',CCR OR CLAST

17AA'   37                          SCF                     ;WORD FULLY LISTED
17AB'   C9                          RET

17AC'                       LPLBRACKET:
17AC'   3E 29                       LD      A,')'
17AE'   18 02                       JR      LPSTRING

17B0'                       LPPTSTR:
17B0'   3E 22                       LD      A,'"'
17B2'                       LPSTRING:
17B2'   F5                          PUSH    AF
17B3'   C5                          PUSH    BC
17B4'   CD 17E1'                    CALL    OUTWORDI
17B7'   D1                          POP     DE
17B8'   CD 0979'                    CALL    TYPEDE          ;DISPLAY STRING
17BB'   42                          LD      B,D
17BC'   4B                          LD      C,E
17BD'   F1                          POP     AF
                                    RSTEMIT                 ;OUTPUT DELIMITER
17BE'   CF              +           RST     008H

17BF'   A7                          AND     A               ;WORD NOT YET FULLY LISTED
17C0'   C9                          RET
                            ;-----------------------------------------------------------------
17C1'                       OUTINDENT:
17C1'   3A 3C15                     LD      A,(LPIACT)
17C4'   A7                          AND     A
17C5'   F8                          RET     M               ;NO NEW LINE & INDENTATION?

17C6'   C5                          PUSH    BC
17C7'   47                          LD      B,A
17C8'   3E 0D                       LD      A,CCR
                                    RSTEMIT
17CA'   CF              +           RST     008H
```

```
17CB'   04                        INC     B
17CC'   05                        DEC     B
17CD'   28 05                     JR      Z,OIQUIT        ;INDENTATION = 0 ?
17CF'                     OILOOP:
17CF'   3E 20                     LD      A,' '
                                  RSTEMIT
17D1'   CF          +             RST     008H
17D2'   10 FB                     DJNZ    OILOOP          ;OUTPUT INDENTATION

17D4'                     OIQUIT:
17D4'   DD 36 15 FF               LD      (IX+LPIACT-MEMBEG),-1    ;NO FURTHER INDENTATION
17D8'   C1                        POP     BC
17D9'   C9                        RET
                          ;-----------------------------------------------------------------
17DA'                     LPNXTWRD:
17DA'   0A                        LD      A,(BC)
17DB'   5F                        LD      E,A
17DC'   03                        INC     BC
17DD'   0A                        LD      A,(BC)
17DE'   57                        LD      D,A
17DF'   03                        INC     BC              ;GET THE NEXT WORD
17E0'   C9                        RET
                          ;-----------------------------------------------------------------
17E1'                     OUTWORDI:
17E1'   CD 17C1'                  CALL    OUTINDENT

17E4'                     OUTWORD:
17E4'   EB                        EX      DE,HL
17E5'   2B                        DEC     HL
17E6'   7E                        LD      A,(HL)
17E7'   CB 7F                     BIT     7,A
17E9'   20 05                     JR      NZ,OWDOXX       ;NOT A NORMAL FORTH-WORD?
17EB'   CD 15E8'                  CALL    PTR2NAME
17EE'   18 0B                     JR      OUTTXT

17F0'                     OWDOXX:
17F0'   EB                        EX      DE,HL
17F1'   CD 15A2'                  CALL    SKOFFS2
17F4'   13                        INC     DE
17F5'   1A                        LD      A,(DE)
17F6'   6F                        LD      L,A
17F7'   13                        INC     DE
17F8'   1A                        LD      A,(DE)
17F9'   67                        LD      H,A
17FA'   19                        ADD     HL,DE           ;POINTER BY NAME

17FB'                     OUTTXT:
17FB'   7E                        LD      A,(HL)
17FC'   E6 7F                     AND     7FH             ;GET CHAR
                                  RSTEMIT
17FE'   CF          +             RST     008H
17FF'   CB 7E                     BIT     7,(HL)
1801'   23                        INC     HL
1802'   28 F7                     JR      Z,OUTTXT        ;NOT FINISHED YET?
1804'   3E 20                     LD      A,' '
                                  RSTEMIT
```

```
1806'   CF          +           RST     008H
1807'   C9                      RET
                        ;-----------------------------------------------------------------
1808'                   ROMTXT:
1808'   E3                      EX      (SP),HL         ;GET POINTER
1809'   CD 17FB'                CALL    OUTTXT
180C'   E3                      EX      (SP),HL         ;SET IT AS RETURN
180D'   C9                      RET
                        ;=================================================================
180E'                   PNTHL:
180E'   11 09B3'                LD      DE,PNT
1811'   D5                      PUSH    DE
1812'   EB                      EX      DE,HL
                                RSTPUSH
1813'   D7          +           RST     010H
1814'   D1                      POP     DE
                        ;-----------------------------------------------------------------
1815'                   EXECDE:
1815'   C5                      PUSH    BC
1816'   CD 04BF'                CALL    NEXTDE
1819'   181B'                   DW      $+2
181B'   181D'                   DW      $+2
181D'   C1                      POP     BC
181E'   C1                      POP     BC
181F'   C9                      RET
                        ;=================================================================
1820'                   TXALL:
1820'   FD E5                   PUSH    IY

1822'   E5                      PUSH    HL
1823'   FD E1                   POP     IY              ;GET ADDRESS

1825'   21 1892'                LD      HL,TXRXQUIT
1828'   E5                      PUSH    HL              ;SET RETURN POINT

1829'   21 E000                 LD      HL,-2000H
182C'   CB 79                   BIT     7,C
182E'   28 02                   JR      Z,TAGOON1       ;LONG STARTER?
1830'   26 FC                   LD      H,-0400H SHR 8
1832'                   TAGOON1:

1832'   13                      INC     DE
1833'   FD 2B                   DEC     IY              ;CORRECT POINTER AND COUNTER
1835'   F3                      DI
1836'   AF                      XOR     A               ;PREPARE
1837'                   TALOOP1:
1837'   06 97                   LD      B,151
1839'                   TADEL1:
1839'   10 FE                   DJNZ    TADEL1          ;LONG STARTER

183B'   D3 FE                   OUT     (IO),A          ;CHANGE LEVEL
183D'   EE 08                   XOR     8

183F'   2C                      INC     L
1840'   20 01                   JR      NZ,TAGOON2
1842'   24                      INC     H
```

```
1843'                          TAGOON2:
1843'   20 F2                          JR      NZ,TALOOP1      ;SEND TITLE

1845'   06 2B                          LD      B,43
1847'                          TADEL2:
1847'   10 FE                          DJNZ    TADEL2          ;SHORT WAIT

1849'   D3 FE                          OUT     (IO),A          ;LEVEL = 0

184B'   69                             LD      L,C             ;GET STARTBYTE

184C'   01 3B08                        LD      BC,8 + (59 SHL 8)
184F'                          TADEL3:
184F'   10 FE                          DJNZ    TADEL3          ;SHORT WAIT

1851'   79                             LD      A,C
1852'   D3 FE                          OUT     (IO),A          ;LEVEL = 1

1854'   06 38                          LD      B,56
1856'   C3 188A'                       JP      TASTART
                               ; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1859'                          TALOOP2:
1859'   79                             LD      A,C             ;GET LEVEL 1
185A'   CB 78                          BIT     7,B             ;SET Z FLAG
185C'                          TADEL4:
185C'   10 FE                          DJNZ    TADEL4          ;SHORT WAIT

185E'   30 04                          JR      NC,TABIT0       ;BIT = 0 ?

1860'   06 3D                          LD      B,61
1862'                          TADEL5:
1862'   10 FE                          DJNZ    TADEL5          ;SHORT WAIT

1864'                          TABIT0:
1864'   D3 FE                          OUT     (IO),A          ;SET LEVEL
1866'   06 3A                          LD      B,58
1868'   C2 1859'                       JP      NZ,TALOOP2      ;FIRST BIT SENT?

186B'   05                             DEC     B               ;CYCLES CORRECTION
186C'   AF                             XOR     A               ;GET LEVEL 0
186D'                          TANEXT:
186D'   CB 15                          RL      L
186F'   C2 185C'                       JP      NZ,TADEL4       ;8 BITS NOT SENT YET?

1872'   1B                             DEC     DE              ;REDUCE COUNT
1873'   FD 23                          INC     IY              ;INCREASE POINTER

1875'   06 2E                          LD      B,46

1877'   3E 7F                          LD      A,7FH
1879'   DB FE                          IN      A,(IO)
187B'   1F                             RRA
187C'   D0                             RET     NC              ;USER CANCEL?

187D'   7A                             LD      A,D
187E'   FE FF                          CP      0FFH
```

```
1880'   D0                      RET     NC              ;CHECKSUM SENT?

1881'   B3                      OR      E
1882'   28 0B                   JR      Z,TAEND         ;ALL BYTES SENT?

1884'   FD 6E 00                LD      L,(IY+0)        ;GET NEXT BYTE
1887'                   TACHECK:
1887'   7C                      LD      A,H
1888'   AD                      XOR     L
1889'   67                      LD      H,A             ;CREATE CHECKSUM

188A'                   TASTART:
188A'   AF                      XOR     A
188B'   37                      SCF                     ;TO BIT COUNT
188C'   C3 186D'                JP      TANEXT

188F'                   TAEND:
188F'   6C                      LD      L,H             ;SEND FINAL CHECKSUM
1890'   18 F5                   JR      TACHECK
                        ;-----------------------------------------------------------------
1892'                   TXRXQUIT:
1892'   FD E1                   POP     IY
1894'   08                      EX      AF,AF'

1895'   06 3B                   LD      B,59
1897'                   TRQDEL6:
1897'   10 FE                   DJNZ    TRQDEL6         ;SHORT WAIT

1899'   AF                      XOR     A
189A'   D3 FE                   OUT     (IO),A          ;LEVEL = 0

189C'   3E 7F                   LD      A,7FH
189E'   DB FE                   IN      A,(IO)
18A0'   1F                      RRA
18A1'   FB                      EI
18A2'   D2 04F0'                JP      NC,BREAK        ;USER CANCEL?

18A5'   08                      EX      AF,AF'
18A6'   C9                      RET
                        ;-----------------------------------------------------------------
18A7'                   RXALL:
18A7'   F3                      DI
18A8'   FD E5                   PUSH    IY

18AA'   E5                      PUSH    HL
18AB'   FD E1                   POP     IY              ;GET POINTER

18AD'   21 1892'                LD      HL,TXRXQUIT
18B0'   E5                      PUSH    HL              ;SET RETURN POINT

18B1'   61                      LD      H,C             ;KEEP START BYTE
18B2'   08                      EX      AF,AF'          ;KEEP READ/VERIFY-FLAG

18B3'   AF                      XOR     A
18B4'   4F                      LD      C,A             ;UPTO LEVEL 0
```

```
18B5'                   RASYNC:
18B5'   C0                      RET     NZ              ;USER-BREAK ?
18B6'                   RALOOP1:
18B6'   2E 00                   LD      L,0
18B8'                   RALOOP2:
18B8'   06 B8                   LD      B,-72
18BA'   CD 1911'                CALL    RXBIT
18BD'   30 F6                   JR      NC,RASYNC       ;CANCELED ?
18BF'   3E DF                   LD      A,-33
18C1'   B8                      CP      B
18C2'   30 F2                   JR      NC,RALOOP1      ;NO SYNC SIGNAL?
18C4'   2C                      INC     L
18C5'   20 F1                   JR      NZ,RALOOP2      ;NO 256 CHARS SYNC ?

18C7'                   RALOOP3:
18C7'   06 CF                   LD      B,-49
18C9'   CD 1915'                CALL    RXLEVEL
18CC'   30 E7                   JR      NC,RASYNC       ;CANCELED ?

18CE'   78                      LD      A,B
18CF'   FE D8                   CP      -40
18D1'   30 F4                   JR      NC,RALOOP3      ;NO SYNC SIGNAL YET?

18D3'   CD 1915'                CALL    RXLEVEL
18D6'   D0                      RET     NC              ;CANCELED ?

18D7'   CD 18FC'                CALL    RXBYTE
18DA'   D0                      RET     NC              ;CANCELED ?

18DB'   3F                      CCF
18DC'   C0                      RET     NZ              ;A WRONG BYTE?
18DD'   18 11                   JR      RASTART
                        ; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
18DF'                   RALOOP:
18DF'   08                      EX      AF,AF'
18E0'   30 05                   JR      NC,RAVERIFY     ;COMPARE ONLY?

18E2'   FD 75 00                LD      (IY+0),L        ;SAVE BYTE
18E5'   18 05                   JR      RAGOON
18E7'                   RAVERIFY:
18E7'   FD 7E 00                LD      A,(IY+0)
18EA'   AD                      XOR     L
18EB'   C0                      RET     NZ              ;DIFFERENT BYTE?

18EC'                   RAGOON:
18EC'   FD 23                   INC     IY              ;INCREMENT POINTER
18EE'   1B                      DEC     DE              ;DECREMENT COUNT
18EF'   08                      EX      AF,AF'

18F0'                   RASTART:
18F0'   CD 18FC'                CALL    RXBYTE
18F3'   D0                      RET     NC              ;CANCELED ?

18F4'   7A                      LD      A,D
18F5'   B3                      OR      E
18F6'   20 E7                   JR      NZ,RALOOP       ;COUNT NOT RECEIVED?
```

```
18F8'   7C                          LD      A,H
18F9'   FE 01                       CP      1               ;SET C IF CHECKSUM OK
18FB'                       RETURN:
18FB'   C9                          RET
                            ;-----------------------------------------------------------------
18FC'                       RXBYTE:
18FC'   2E 01                       LD      L,1             ;FOR BIT COUNT

18FE'                       RB8LOOP:
18FE'   06 C7                       LD      B,-57
1900'   CD 1911'                    CALL    RXBIT
1903'   D0                          RET     NC              ;CANCELED ?
1904'   3E E2                       LD      A,-30
1906'   B8                          CP      B               ;LONGER TIME = BIT 1
1907'   CB 15                       RL      L
1909'   D2 18FE'                    JP      NC,RB8LOOP      ;NOT YET 8 BITS?

190C'   7C                          LD      A,H
190D'   AD                          XOR     L
190E'   67                          LD      H,A             ;BUILD CHECKSUM

190F'   37                          SCF                     ;BYTE RECEIVED
1910'   C9                          RET
                            ;-----------------------------------------------------------------
1911'                       RXBIT:
1911'   CD 1915'                    CALL    RXLEVEL
1914'   D0                          RET     NC              ;CANCELED ?

1915'                       RXLEVEL:
1915'   3E 14                       LD      A,20
1917'                       RBDELAY:
1917'   3D                          DEC     A
1918'   20 FD                       JR      NZ,RBDELAY      ;SHORT WAIT

191A'   A7                          AND     A               ;CLEAR CHAR
191B'                       RBLOOP:
191B'   04                          INC     B
191C'   C8                          RET     Z               ;TIMEOUT ?

191D'   3E 7F                       LD      A,7FH
191F'   DB FE                       IN      A,(IO)
1921'   1F                          RRA
1922'   D0                          RET     NC              ;USER-CANCEL?

1923'   A9                          XOR     C
1924'   E6 10                       AND     020H SHR 1
1926'   28 F3                       JR      Z,RBLOOP        ;SAME LEVEL ?

1928'   79                          LD      A,C
1929'   2F                          CPL
192A'   4F                          LD      C,A             ;TOGGLE LEVEL

192B'   37                          SCF                     ;ALL OK
192C'   C9                          RET
                            ;=================================================================
```

```
0000                        FFLAG   EQU     0       ;00/FF = DICTIONARY / BINARY FILE
0001                        FNLEN   EQU     1       ;NAME LENGTH
                            ;               2       ;FILE NAME
000B                        FLEN    EQU     11      ;NUMBER OF BYTES
000D                        FSTART  EQU     13      ;START ADDRESS
000F                        FDICT   EQU     15      ;DICTIONARY
0011                        FCURR   EQU     17      ;VCURRENT
                            ;               19      ;VCONTEXT
                            ;               21      ;VOCLNK
                            ;               23      ;STKBOT

0019                        FSIZE   EQU     25      ;SIZE OF THIS BLOCK
                            ;=================================================================
192D'   53 41 56 C5                 DB      'SAV','E' OR CLAST
1931'   166F'                       DW      LIST-1
1933'   04                          DB      4
1934'                       SAVE:
1934'   0EC3'                       DW      DOCOL
1936'   1A10' 1A4F'                 DW      FILEFHEAD,DOSAVE
193A'   04B6'                       DW      SEMIS
                            ;=================================================================
193C'   42 53 41 56                 DB      'BSAV','E' OR CLAST
1940'   C5
1941'   1933'                       DW      SAVE-1
1943'   05                          DB      5
1944'                       BSAVE:
1944'   0EC3'                       DW      DOCOL
1946'   1A3D' 1A4F'                 DW      FILEBHEAD,DOSAVE
194A'   04B6'                       DW      SEMIS
                            ;=================================================================
194C'   42 4C 4F 41                 DB      'BLOA','D' OR CLAST
1950'   C4
1951'   1943'                       DW      BSAVE-1
1953'   05                          DB      5
1954'                       BLOAD:
1954'   0EC3'                       DW      DOCOL
1956'   1A3D' 1A74'                 DW      FILEBHEAD,READHEADER,DOBLOAD
195A'   1AB8'
195C'   04B6'                       DW      SEMIS
                            ;=================================================================
195E'   56 45 52 49                 DB      'VERIF','Y' OR CLAST
1962'   46 D9
1964'   1953'                       DW      BLOAD-1
1966'   06                          DB      6
1967'                       VERIFY:
1967'   0EC3'                       DW      DOCOL
1969'   1A10'                       DW      FILEFHEAD
196B'   1271' 000F                  DW      DOELSE,DOVERIFY-$-1
                            ;=================================================================
196F'   42 56 45 52                 DB      'BVERIF','Y' OR CLAST
1973'   49 46 D9
1976'   1966'                       DW      VERIFY-1
1978'   07                          DB      7
1979'                       BVERIFY:
1979'   0EC3'                       DW      DOCOL
197B'   1A3D'                       DW      FILEBHEAD
```

```
197D'                      DOVERIFY:
197D'   1A74' 1ABE'                DW      READHEADER,DOBVERIFY
1981'   04B6'                      DW      SEMIS
                           ;============================================================
1983'   4C 4F 41 C4                DB      'LOA','D' OR CLAST
1987'   1978'                      DW      BVERIFY-1
1989'   04                         DB      4
198A'                      LOAD:
198A'   0EC3'                      DW      DOCOL
198C'   1A10'                      DW      FILEFHEAD
198E'   1A0E'                      DW      SEMICODE

1990'   2A 3C37                    LD      HL,(STKBOT)
1993'   22 230E                    LD      (FPADMEM+FSTART),HL     ;START

1996'   EB                         EX      DE,HL
1997'   21 FFCC                    LD      HL,-52
199A'   39                         ADD     HL,SP
199B'   A7                         AND     A
199C'   ED 52                      SBC     HL,DE
199E'   22 230C                    LD      (FPADMEM+FLEN),HL       ;SIZE OF MEMORY FREE

19A1'   CD 04B9'                   CALL    NEXT
19A4'   1A74' 1AB8'                DW      READHEADER,DOBLOAD
19A8'   1A0E'                      DW      SEMICODE

19AA'   ED 4B 3C37                 LD      BC,(STKBOT)

19AE'   21 3C50                    LD      HL,FREEMEM-1
19B1'   22 2701                    LD      (PADMEM+RDONAME),HL
19B4'   23                         INC     HL
19B5'   22 2709                    LD      (PADMEM+RDOEND),HL      ;PREPARE ADJUSTMENT

19B8'   2A 2325                    LD      HL,(FPADMEM+FSIZE+FLEN)
19BB'   09                         ADD     HL,BC
19BC'   22 3C37                    LD      (STKBOT),HL     ;RESERVE MEMORY

19BF'   21 C3AF                    LD      HL,-FREEMEM
19C2'   09                         ADD     HL,BC
19C3'   22 270B                    LD      (PADMEM+RDDLEN),HL
19C6'   ED 5B 2329                 LD      DE,(FPADMEM+FSIZE+FDICT)
19CA'   19                         ADD     HL,DE
19CB'   ED 5B 3C4C                 LD      DE,(FORTH+2+RAMVAR-ROMVAR)
19CF'   22 3C4C                    LD      (FORTH+2+RAMVAR-ROMVAR),HL      ;NEW END

19D2'   C5                         PUSH    BC
19D3'   D5                         PUSH    DE
19D4'   ED 73 270D                 LD      (PADMEM+RDNNAME),SP
19D8'   CD 1504'                   CALL    CORRDICT        ;LINK LOADED
19DB'   C1                         POP     BC
19DC'   E1                         POP     HL
19DD'                      LDNLOOP:
19DD'   CB 7E                      BIT     7,(HL)
19DF'   23                         INC     HL
19E0'   28 FB                      JR      Z,LDNLOOP       ;SKIP NAME
19E2'   23                         INC     HL
```

```
19E3'   23                          INC     HL
19E4'   71                          LD      (HL),C
19E5'   23                          INC     HL
19E6'   70                          LD      (HL),B          ;SAVE LENGTH OF DICT.

19E7'   2A 3C37                     LD      HL,(STKBOT)
19EA'   01 000C                     LD      BC,SAFETY
19ED'   09                          ADD     HL,BC
19EE'   22 3C3B                     LD      (SPARE),HL      ;ADJUST PARAMETER STACK
19F1'   FD E9                       JP      (IY)
                            ;=================================================================
19F3'                       FILENAME:
19F3'   0EC3'                       DW      DOCOL
19F5'   104B'                       DW      GETBYTE
19F7'   20                          DB      ' '
19F8'   05AB'                       DW      WORD
19FA'   1A0E'                       DW      SEMICODE        ;GET NAME
19FC'   CD 0F2E'                    CALL    LINKHERE
                                    RSTPULL
19FF'   DF                  +       RST     018H
1A00'   3E 20                       LD      A,' '
1A02'   12                          LD      (DE),A          ;REPLACE NAME LENGTH BY ' '

1A03'   11 270C                     LD      DE,PADMEM+FLEN
1A06'   21 27FF                     LD      HL,SCRMEND-1
1A09'   CD 07FA'                    CALL    BLANKS          ;CLEAR BUFFER
1A0C'   FD E9                       JP      (IY)
                            ;=================================================================
1A0E'                       SEMICODE:
1A0E'   18FB'                       DW      RETURN
                            ;=================================================================
1A10'                       FILEFHEAD:
1A10'   0EC3'                       DW      DOCOL
1A12'   19F3'                       DW      FILENAME
1A14'   1A0E'                       DW      SEMICODE
1A16'   AF                          XOR     A
1A17'   32 2301                     LD      (FPADMEM+FFLAG),A
1A1A'   21 3C51                     LD      HL,FREEMEM
1A1D'   22 230E                     LD      (FPADMEM+FSTART),HL
1A20'   EB                          EX      DE,HL
1A21'   2A 3C37                     LD      HL,(STKBOT)
1A24'   A7                          AND     A
1A25'   ED 52                       SBC     HL,DE
1A27'   22 230C                     LD      (FPADMEM+FLEN),HL
1A2A'   2A 3C4C                     LD      HL,(FORTH+2+RAMVAR-ROMVAR)
1A2D'   22 2310                     LD      (FPADMEM+FDICT),HL
1A30'   21 3C31                     LD      HL,VCURRENT
1A33'   11 2312                     LD      DE,FPADMEM+FCURR
1A36'   01 0008                     LD      BC,8
1A39'   ED B0                       LDIR                    ;PREPARE HEADER
1A3B'   FD E9                       JP      (IY)
                            ;=================================================================
1A3D'                       FILEBHEAD:
1A3D'   0EC3'                       DW      DOCOL
1A3F'   19F3'                       DW      FILENAME
1A41'   1011' 230C                  DW      GETWORD,FPADMEM+FLEN,EXCLAM
```

```
1A45'   08C1'
1A47'   1011' 230E              DW      GETWORD,FPADMEM+FSTART,EXCLAM
1A4B'   08C1'
1A4D'   04B6'                   DW      SEMIS
                        ;=================================================================
1A4F'                   DOSAVE:
1A4F'   1A51'                   DW      $+2

1A51'   3A 2302                 LD      A,(FPADMEM+FNLEN)
1A54'   A7                      AND     A
1A55'   28 5F                   JR      Z,RXERROR       ;NO NAME ?
1A57'   2A 230C                 LD      HL,(FPADMEM+FLEN)
1A5A'   7C                      LD      A,H
1A5B'   B5                      OR      L
1A5C'   28 58                   JR      Z,RXERROR       ;LENGTH = 0 ?
1A5E'   E5                      PUSH    HL

1A5F'   11 0019                 LD      DE,25
1A62'   21 2301                 LD      HL,FPADMEM+FFLAG
1A65'   4A                      LD      C,D
1A66'   CD 1820'                CALL    TXALL           ;SEND HEADER

1A69'   D1                      POP     DE
1A6A'   2A 230E                 LD      HL,(FPADMEM+FSTART)
1A6D'   0E FF                   LD      C,-1
1A6F'   CD 1820'                CALL    TXALL           ;SEND DATA
1A72'   FD E9                   JP      (IY)
                        ;-----------------------------------------------------------------
1A74'                   READHEADER:
1A74'   1A76'                   DW      $+2

1A76'                   RHLOOP:
1A76'   11 0019                 LD      DE,25
1A79'   21 231A                 LD      HL,FPADMEM+FSIZE+FFLAG
1A7C'   4A                      LD      C,D
1A7D'   37                      SCF
1A7E'   CD 18A7'                CALL    RXALL           ;READ HEADER
1A81'   30 F3                   JR      NC,RHLOOP       ;NOT YET OK ?

1A83'   11 231A                 LD      DE,FPADMEM+FSIZE+FFLAG
1A86'   1A                      LD      A,(DE)
1A87'   A7                      AND     A
1A88'   20 0B                   JR      NZ,RHBINARY     ;BINARY FILE?

1A8A'   CD 1808'                CALL    ROMTXT
1A8D'   0D 44 69 63             DB      CCR,'Dict',':' OR CLAST
1A91'   74 BA
1A93'   18 0A                   JR      RHCHECK

1A95'                   RHBINARY:
1A95'   CD 1808'                CALL    ROMTXT
1A98'   0D 42 79 74             DB      CCR,'Bytes',':' OR CLAST
1A9C'   65 73 BA

1A9F'                   RHCHECK:
1A9F'   21 2301                 LD      HL,FPADMEM+FFLAG
```

```
1AA2'   01 0B0B                 LD      BC,11 + (11 SHL 8)
1AA5'   18 02                   JR      RHCSTART

1AA7'                   RHCLOOP:
1AA7'   1A                      LD      A,(DE)
                                RSTEMIT                 ;DISPLAY NAME
1AA8'   CF              +       RST     008H
1AA9'                   RHCSTART:
1AA9'   1A                      LD      A,(DE)
1AAA'   BE                      CP      (HL)
1AAB'   20 01                   JR      NZ,RHCNEXT      ;DIFFERENT CHAR?
1AAD'   0D                      DEC     C
1AAE'                   RHCNEXT:
1AAE'   23                      INC     HL
1AAF'   13                      INC     DE
1AB0'   10 F5                   DJNZ    RHCLOOP         ;NOT YET ALL CHARS?

1AB2'   20 C2                   JR      NZ,RHLOOP       ;DIFFERENT NAME ?
1AB4'   FD E9                   JP      (IY)
                        ;-----------------------------------------------------------------
1AB6'                   RXERROR:
                                RSTERR  ERRREAD
1AB6'   E7              +       RST     020H
1AB7'   0A              +       DB      ERRREAD
                        ;-----------------------------------------------------------------
1AB8'                   DOBLOAD:
1AB8'   1ABA'                   DW      $+2

1ABA'   06 FF                   LD      B,-1            ;READ
1ABC'   18 12                   JR      DOBREAD
                        ;-----------------------------------------------------------------
1ABE'                   DOBVERIFY:
1ABE'   1AC0'                   DW      $+2

1AC0'   21 2312                 LD      HL,FPADMEM+FCURR
1AC3'   11 232B                 LD      DE,FPADMEM+FSIZE+FCURR
1AC6'   06 08                   LD      B,8
1AC8'                   DBVLOOP:
1AC8'   1A                      LD      A,(DE)
1AC9'   13                      INC     DE
1ACA'   BE                      CP      (HL)
1ACB'   23                      INC     HL
1ACC'   20 E8                   JR      NZ,RXERROR
1ACE'   10 F8                   DJNZ    DBVLOOP         ;COMPARE VARIABLES

1AD0'                   DOBREAD:
1AD0'   2A 230C                 LD      HL,(FPADMEM+FLEN)
1AD3'   ED 5B 2325              LD      DE,(FPADMEM+FSIZE+FLEN)
1AD7'   7C                      LD      A,H
1AD8'   B5                      OR      L
1AD9'   28 04                   JR      Z,DBRGOON1      ;DO NO LENGTH TEST?
1ADB'   ED 52                   SBC     HL,DE
1ADD'   38 D7                   JR      C,RXERROR

1ADF'                   DBRGOON1:
1ADF'   2A 230E                 LD      HL,(FPADMEM+FSTART)
```

```
1AE2'   7C                              LD      A,H
1AE3'   B5                              OR      L
1AE4'   20 03                           JR      NZ,DBRGOON2     ;USE A START ADDRESS?
1AE6'   2A 2327                         LD      HL,(FPADMEM+FSIZE+FSTART)

1AE9'                           DBRGOON2:
1AE9'   0E FF                           LD      C,-1
1AEB'   CB 18                           RR      B               ;GET READ/VERIFY-FLAG
1AED'   CD 18A7'                        CALL    RXALL           ;READ DATA
1AF0'   30 C4                           JR      NC,RXERROR      ;CANCELED ?
1AF2'   FD E9                           JP      (IY)
                                ;================================================================
0000                            FEXP1   EQU     0       ;EXPONENT UPPER NUMBER / RESULT
0001                            FEXP2   EQU     1       ;EXPONENT LOWER NUMBER
0002                            FSGN    EQU     2       ;SIGN 7=BELOW 6=ABOVE
0003                            FACCU   EQU     3       ;ACCUMULATOR
0007                            FQUO    EQU     7       ;QUOTIENT
0010                            FDIVOR  EQU     16      ;DIVISOR
                                ;================================================================
1AF4'                           FINIT:
1AF4'   01 3C0F                         LD      BC,FPWS+FDIVOR-1
1AF7'   AF                              XOR     A
1AF8'                           FICLEAR:
1AF8'   02                              LD      (BC),A
1AF9'   0D                              DEC     C               ;(NOT CLEAN !!!)
1AFA'   20 FC                           JR      NZ,FICLEAR      ;CLEAR BUFFER

1AFC'   2A 3C3B                         LD      HL,(SPARE)
1AFF'   11 FFFC                         LD      DE,-4
1B02'   2B                              DEC     HL
1B03'   4E                              LD      C,(HL)          ;KEEP EXPONENT UPPER NUMBER
1B04'   77                              LD      (HL),A          ;  AND CLEAR IT
1B05'   19                              ADD     HL,DE
1B06'   23                              INC     HL
1B07'   22 3C3B                         LD      (SPARE),HL      ;REMOVE "TOS"
1B0A'   2B                              DEC     HL
1B0B'   46                              LD      B,(HL)          ;KEEP EXPONENT LOWER NUMBER
1B0C'   77                              LD      (HL),A          ;  AND CLEAR IT
1B0D'   79                              LD      A,C
1B0E'   0F                              RRCA
1B0F'   A8                              XOR     B
1B10'   E6 7F                           AND     NOT FSIGN
1B12'   A8                              XOR     B
1B13'   32 3C02                         LD      (FPWS+FSGN),A   ;KEEP THE SIGN
1B16'   CB B8                           RES     7,B
1B18'   CB B9                           RES     7,C
1B1A'   ED 43 3C00                      LD      (FPWS+FEXP1),BC ;SAVE EXPONENTS
1B1E'   23                              INC     HL
1B1F'   EB                              EX      DE,HL           ;POINT UPPER NUMBER
1B20'   19                              ADD     HL,DE           ;POINT LOWER NUMBER
1B21'   C9                              RET
                                ;----------------------------------------------------------------
1B22'                           FADJUST:
1B22'   3E 09                           LD      A,9
1B24'   B8                              CP      B
1B25'   30 01                           JR      NC,FADJLP1      ;LIMIT EXPONENT DIFFERENCES
```

```
1B27'   47                      LD      B,A

1B28'                   FADJLP1:
1B28'   0E 04                   LD      C,4
1B2A'   23                      INC     HL
1B2B'   23                      INC     HL
1B2C'   23                      INC     HL
1B2D'   AF                      XOR     A
1B2E'                   FADJLP2:
1B2E'   ED 67                   RRD
1B30'   2B                      DEC     HL
1B31'   0D                      DEC     C
1B32'   20 FA                   JR      NZ,FADJLP2      ;DIVIDE SMALLER NUMBERS
1B34'   23                      INC     HL
1B35'   10 F1                   DJNZ    FADJLP1         ;BY APPROXIMATION

1B37'   C6 FB                   ADD     A,-5            ;WAS LAST POINT >= 5?
1B39'   E5                      PUSH    HL
1B3A'                   FADJLP3:
1B3A'   7E                      LD      A,(HL)
1B3B'   88                      ADC     A,B
1B3C'   27                      DAA
1B3D'   77                      LD      (HL),A
1B3E'   23                      INC     HL
1B3F'   38 F9                   JR      C,FADJLP3       ;ROUND
1B41'   E1                      POP     HL
1B42'   C9                      RET
                        ;-----------------------------------------------------------------
1B43'                   FNEG:
1B43'   C5                      PUSH    BC
1B44'   E5                      PUSH    HL
1B45'   06 04                   LD      B,4
1B47'   A7                      AND     A
1B48'                   FNLOOP:
1B48'   3E 00                   LD      A,0
1B4A'   9E                      SBC     A,(HL)
1B4B'   27                      DAA
1B4C'   77                      LD      (HL),A
1B4D'   23                      INC     HL
1B4E'   10 F8                   DJNZ    FNLOOP          ;NEGATE ALL
1B50'   E1                      POP     HL
1B51'   C1                      POP     BC
1B52'   C9                      RET
                        ;-----------------------------------------------------------------
1B53'                   FADDITION:
1B53'   0E 01                   LD      C,1             ;MULTIPLICATOR 1

1B55'                   FMULADD:
1B55'   E5                      PUSH    HL
1B56'   D5                      PUSH    DE
1B57'   C5                      PUSH    BC
1B58'   79                      LD      A,C
1B59'   E6 0F                   AND     0FH
1B5B'   47                      LD      B,A
1B5C'   A9                      XOR     C
1B5D'   4F                      LD      C,A
```

```
1B5E'   0F                      RRCA
1B5F'   0F                      RRCA
1B60'   81                      ADD     A,C
1B61'   0F                      RRCA
1B62'   80                      ADD     A,B
1B63'   4F                      LD      C,A             ;CONVERT BCD TO BINARY

1B64'   06 04                   LD      B,4
1B66'   AF                      XOR     A
1B67'                   FMLOOP1:
1B67'   C5                      PUSH    BC
1B68'   D5                      PUSH    DE
1B69'   E5                      PUSH    HL
1B6A'   86                      ADD     A,(HL)
1B6B'   27                      DAA
1B6C'   6F                      LD      L,A
1B6D'   1A                      LD      A,(DE)
1B6E'   26 00                   LD      H,0
1B70'   54                      LD      D,H
1B71'   CB 14                   RL      H               ;OVERFLOW FROM ADDITION
1B73'   A7                      AND     A
1B74'   28 1B                   JR      Z,FMNEXT        ;DIGIT = 0?
1B76'   5F                      LD      E,A
1B77'                   FMLOOP2:
1B77'   CB 39                   SRL     C
1B79'   30 08                   JR      NC,FMNOADD      ;MULTIPLICATOR-BIT = 0 ?
1B7B'   7D                      LD      A,L
1B7C'   83                      ADD     A,E
1B7D'   27                      DAA
1B7E'   6F                      LD      L,A
1B7F'   7C                      LD      A,H
1B80'   8A                      ADC     A,D
1B81'   27                      DAA
1B82'   67                      LD      H,A             ;ADDITION
1B83'                   FMNOADD:
1B83'   0C                      INC     C
1B84'   0D                      DEC     C
1B85'   28 0A                   JR      Z,FMNEXT        ;MULTIPLICATOR = 0 ?
1B87'   7B                      LD      A,E
1B88'   87                      ADD     A,A
1B89'   27                      DAA
1B8A'   5F                      LD      E,A
1B8B'   7A                      LD      A,D
1B8C'   8F                      ADC     A,A
1B8D'   27                      DAA
1B8E'   57                      LD      D,A             ;SHIFT RESULT
1B8F'   18 E6                   JR      FMLOOP2         ;AGAIN

1B91'                   FMNEXT:
1B91'   EB                      EX      DE,HL
1B92'   E1                      POP     HL
1B93'   73                      LD      (HL),E
1B94'   7A                      LD      A,D
1B95'   D1                      POP     DE
1B96'   C1                      POP     BC
1B97'   13                      INC     DE
```

```
1B98'   23                      INC     HL
1B99'   10 CC                   DJNZ    FMLOOP1         ;NOT YET ALL BYTES?

1B9B'   C1                      POP     BC
1B9C'   D1                      POP     DE
1B9D'   E1                      POP     HL
1B9E'   C9                      RET
                        ;=================================================================
1B9F'   46 AD                   DB      'F','-' OR CLAST
1BA1'   1989'                   DW      LOAD-1
1BA3'   02                      DB      2
1BA4'                   FMINUS:
1BA4'   0EC3'                   DW      DOCOL
1BA6'   1D0F'                   DW      FNEGATE
1BA8'   1A0E'                   DW      SEMICODE
1BAA'   18 07                   JR      FADDSUB
                        ;=================================================================
1BAC'   46 AB                   DB      'F','+' OR CLAST
1BAE'   1BA3'                   DW      FMINUS-1
1BB0'   02                      DB      2
1BB1'                   FPLUS:
1BB1'   1BB3'                   DW      FADDSUB

1BB3'                   FADDSUB:
1BB3'   CD 1AF4'                CALL    FINIT           ;PREPARE FOR PROCESSING

1BB6'   79                      LD      A,C
1BB7'   90                      SUB     B
1BB8'   F5                      PUSH    AF
1BB9'   30 06                   JR      NC,FASGOON1     ;EXPONENT BELOW <= UP?
1BBB'   EB                      EX      DE,HL
1BBC'   ED 44                   NEG
1BBE'   DD 70 00                LD      (IX+FPWS+FEXP1-MEMBEG),B        ;EXCHANGE NUMBERS
1BC1'                   FASGOON1:
1BC1'   47                      LD      B,A
1BC2'   C4 1B22'                CALL    NZ,FADJUST      ;ADJUST OTHER NUMBER IF NEEDED
1BC5'   F1                      POP     AF
1BC6'   30 01                   JR      NC,FASGOON2     ;EXPONENT BELOW <= UP?
1BC8'   EB                      EX      DE,HL
1BC9'                   FASGOON2:

1BC9'   06 02                   LD      B,2
1BCB'   DD 4E 02                LD      C,(IX+FPWS+FSGN-MEMBEG)
1BCE'                   FASLP1:
1BCE'   CB 11                   RL      C
1BD0'   DC 1B43'                CALL    C,FNEG
1BD3'   EB                      EX      DE,HL
1BD4'   10 F8                   DJNZ    FASLP1          ;NEGATE WHEN NEEDED

1BD6'   CD 1B53'                CALL    FADDITION
1BD9'   1B                      DEC     DE
1BDA'   1A                      LD      A,(DE)
1BDB'   C6 68                   ADD     A,-98H
1BDD'   CB 18                   RR      B
1BDF'   DD 70 02                LD      (IX+FPWS+FSGN-MEMBEG),B ;KEEP THE NEW SIGN
1BE2'   C4 1B43'                CALL    NZ,FNEG         ;NEGATE ON NEED
```

```
1BE5'                         FASLP2:
1BE5'   1A                            LD      A,(DE)
1BE6'   A7                            AND     A
1BE7'   20 19                         JR      NZ,FASGOON3     ;OBERSTE STELLEN <> 0 ?
1BE9'   DD 35 00                      DEC     (IX+FPWS+FEXP1-MEMBEG)
1BEC'   DD 35 00                      DEC     (IX+FPWS+FEXP1-MEMBEG)  ;ADJUST EXPONENT
1BEF'   D5                            PUSH    DE
1BF0'   62                            LD      H,D
1BF1'   6B                            LD      L,E
1BF2'   2B                            DEC     HL
1BF3'   01 03FF                       LD      BC,255+(3 SHL 8)        ;C LOADED FOR "LDD"
1BF6'                         FASLP3:
1BF6'   B6                            OR      (HL)
1BF7'   ED A8                         LDD
1BF9'   10 FB                         DJNZ    FASLP3          ;SHIFT VALUE
1BFB'   EB                            EX      DE,HL
1BFC'   70                            LD      (HL),B
1BFD'   D1                            POP     DE
1BFE'   20 E5                         JR      NZ,FASLP2       ;NUMBER <> 0 ?
1C00'   FD E9                         JP      (IY)

1C02'                         FASGOON3:
1C02'   54                            LD      D,H
1C03'   5D                            LD      E,L             ;NUMBER NOT YET SHIFTED
                              ; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1C04'                         FCORR:
1C04'   D5                            PUSH    DE
1C05'   01 0004                       LD      BC,4
1C08'   ED B0                         LDIR                    ;SHIFT NUMBER
1C0A'   E1                            POP     HL
1C0B'   1B                            DEC     DE
1C0C'                         FCLP:
1C0C'   1A                            LD      A,(DE)
1C0D'   A7                            AND     A
1C0E'   28 11                         JR      Z,FCQUIT        ;NUMBER = 0 ?

1C10'   FE 10                         CP      10H
1C12'   9F                            SBC     A,A
1C13'   3C                            INC     A
1C14'   3C                            INC     A
1C15'   47                            LD      B,A
1C16'   DD 86 00                      ADD     A,(IX+FPWS+FEXP1-MEMBEG)
1C19'   32 3C00                       LD      (FPWS+FEXP1),A  ;ADJUST EXPONENT

1C1C'   CD 1B22'                      CALL    FADJUST
1C1F'   18 EB                         JR      FCLP

1C21'                         FCQUIT:
1C21'   3A 3C00                       LD      A,(FPWS+FEXP1)
1C24'   3D                            DEC     A
1C25'   FE BF                         CP      -FEOFFS-1
1C27'   3C                            INC     A
1C28'   30 13                         JR      NC,FLT0         ;NUMBER TOO SMALL?
1C2A'   FE 80                         CP      +FEOFFS+64
1C2C'   30 0D                         JR      NC,FLTERR       ;NUMBER TOO BIG?
```

```
1C2E'   47                      LD      B,A
1C2F'   3A 3C02                 LD      A,(FPWS+FSGN)
1C32'   4F                      LD      C,A
1C33'   17                      RLA
1C34'   A9                      XOR     C
1C35'   E6 80                   AND     FSIGN
1C37'   A8                      XOR     B
1C38'   12                      LD      (DE),A          ;SIGN AND EXPONENT
1C39'   FD E9                   JP      (IY)

1C3B'                   FLTERR:
                                RSTERR  ERRFLT
1C3B'   E7          +           RST     020H
1C3C'   08          +           DB      ERRFLT

1C3D'                   FLT0:
1C3D'   01 0400                 LD      BC,0+(4 SHL 8)
1C40'                   FLT0LP:
1C40'   71                      LD      (HL),C
1C41'   23                      INC     HL
1C42'   10 FC                   DJNZ    FLT0LP          ;SET NUMBER TO 0
1C44'   FD E9                   JP      (IY)
                        ;=================================================================
1C46'   46 AA                   DB      'F','*' OR CLAST
1C48'   1BB0'                   DW      FPLUS-1
1C4A'   02                      DB      2
1C4B'                   FMUL:
1C4B'   1C4D'                   DW      $+2

1C4D'   CD 1AF4'                CALL    FINIT           ;PREPARE FOR PROCESSING

1C50'   AF                      XOR     A
1C51'   B8                      CP      B
1C52'   9F                      SBC     A,A
1C53'   A1                      AND     C
1C54'   28 E7                   JR      Z,FLT0          ;IS ONE OF THE TWO = 0?

1C56'   E5                      PUSH    HL
1C57'   01 3C02                 LD      BC,FPWS+FACCU-1
1C5A'   C5                      PUSH    BC
1C5B'   06 03                   LD      B,3
1C5D'                   FMLOOP:
1C5D'   4E                      LD      C,(HL)
1C5E'   23                      INC     HL
1C5F'   E3                      EX      (SP),HL
1C60'   23                      INC     HL
1C61'   CD 1B55'                CALL    FMULADD
1C64'   E3                      EX      (SP),HL
1C65'   10 F6                   DJNZ    FMLOOP          ;MULTIPLY ALL DIGITS IN THE DOUBLE

1C67'   ED 4B 3C00              LD      BC,(FPWS+FEXP1)
1C6B'   78                      LD      A,B
1C6C'   81                      ADD     A,C
1C6D'   D6 42                   SUB     FEOFFS+2
1C6F'   32 3C00                 LD      (FPWS+FEXP1),A  ;CALCULATE EXPONENTS
1C72'   E1                      POP     HL
```

```
1C73'   D1                      POP     DE
1C74'   18 8E                   JR      FCORR
                        ;===================================================================
1C76'   46 AF                   DB      'F','/' OR CLAST
1C78'   1C4A'                   DW      FMUL-1
1C7A'   02                      DB      2
1C7B'                   FDIV:
1C7B'   1C7D'                   DW      $+2

1C7D'   CD 1AF4'                CALL    FINIT           ;PREPARE FOR PROCESSING

1C80'   AF                      XOR     A
1C81'   B8                      CP      B
1C82'   28 B9                   JR      Z,FLT0          ;DIVIDEND = 0 ?

1C84'   B9                      CP      C
1C85'   28 B4                   JR      Z,FLTERR        ;DIVISOR = 0 ?

1C87'   13                      INC     DE
1C88'   13                      INC     DE
1C89'   1A                      LD      A,(DE)
1C8A'   1B                      DEC     DE
1C8B'   1B                      DEC     DE
1C8C'   C6 01                   ADD     A,1
1C8E'   27                      DAA
1C8F'   08                      EX      AF,AF'          ;TEST ON 0.99????E??

1C90'   EB                      EX      DE,HL
1C91'   CD 1B43'                CALL    FNEG            ;NEGATE UPPER NUMBER FOR SUBTR.
1C94'   EB                      EX      DE,HL

1C95'   E5                      PUSH    HL
1C96'   11 3C10                 LD      DE,FPWS+FDIVOR
1C99'   01 0004                 LD      BC,4
1C9C'   ED B0                   LDIR                    ;STORE LOWER NUMBER
1C9E'   EB                      EX      DE,HL
1C9F'   2B                      DEC     HL
1CA0'   06 05                   LD      B,5             ;DIVISOR DIGITS COUNT
1CA2'                   FDLOOP1:
1CA2'   D5                      PUSH    DE
1CA3'   7E                      LD      A,(HL)
1CA4'   2B                      DEC     HL
1CA5'   5E                      LD      E,(HL)
1CA6'   08                      EX      AF,AF'
1CA7'   4F                      LD      C,A
1CA8'   08                      EX      AF,AF'
1CA9'   0C                      INC     C
1CAA'   0D                      DEC     C
1CAB'   20 03                   JR      NZ,FDGOON1      ;WAS NUMBER < 0.990000EXX ?
1CAD'   5F                      LD      E,A
1CAE'   18 1B                   JR      FDGOON2

1CB0'                   FDGOON1:
1CB0'   C5                      PUSH    BC
1CB1'   06 02                   LD      B,2             ;2 DIGIT PER BYTE
1CB3'                   FDLOOP2:
```

```
1CB3'   16 10               LD      D,10H
1CB5'                   FDLOOP3:
1CB5'   CB 23               SLA     E
1CB7'   17                  RLA
1CB8'   CB 12               RL      D
1CBA'   30 F9               JR      NC,FDLOOP3      ;SHIFT D-A-E BY 1 DIGIT
1CBC'   14                  INC     D
1CBD'                   FDLOOP4:
1CBD'   91                  SUB     C
1CBE'   27                  DAA
1CBF'   1C                  INC     E
1CC0'   30 FB               JR      NC,FDLOOP4
1CC2'   15                  DEC     D
1CC3'   20 F8               JR      NZ,FDLOOP4      ;PARTIAL DIVISION BY SUBTRACTION
1CC5'   81                  ADD     A,C
1CC6'   27                  DAA
1CC7'   1D                  DEC     E
1CC8'   10 E9               DJNZ    FDLOOP2         ;CALCULATE SINGLE QUOTIENTS

1CCA'   C1                  POP     BC
1CCB'                   FDGOON2:
1CCB'   4B                  LD      C,E
1CCC'   D1                  POP     DE
1CCD'   0C                  INC     C
1CCE'   0D                  DEC     C
1CCF'   28 17               JR      Z,FDNEXT        ;SINGLE QUOTIENT = 0?

1CD1'   E5                  PUSH    HL
1CD2'   2B                  DEC     HL
1CD3'   2B                  DEC     HL
1CD4'   CD 1B55'            CALL    FMULADD         ;A SUBTRACTION
1CD7'   D5                  PUSH    DE
1CD8'   11 FFFB             LD      DE,FQUO-FDIVOR+4
1CDB'   19                  ADD     HL,DE           ;ALIGN POINTER
1CDC'   11 3C03             LD      DE,FPWS+FACCU
1CDF'   79                  LD      A,C
1CE0'   12                  LD      (DE),A
1CE1'   CD 1B53'            CALL    FADDITION       ;ACCUMULATE ON QUOTIENT
1CE4'   D1                  POP     DE
1CE5'   E1                  POP     HL
1CE6'   23                  INC     HL
1CE7'   04                  INC     B
1CE8'                   FDNEXT:
1CE8'   10 B8               DJNZ    FDLOOP1         ;AND ONE MORE ROUND ...

1CEA'   2A 3C00             LD      HL,(FPWS+FEXP1)
1CED'   7C                  LD      A,H
1CEE'   95                  SUB     L
1CEF'   C6 40               ADD     A,FEOFFS
1CF1'   21 3C08             LD      HL,FPWS+FQUO+1
1CF4'   47                  LD      B,A
1CF5'   3A 3C0B             LD      A,(FPWS+FQUO+4)
1CF8'   A7                  AND     A
1CF9'   20 03               JR      NZ,FDGOON3
1CFB'   05                  DEC     B
1CFC'   05                  DEC     B
```

```
1CFD'   2B                      DEC     HL              ;EXPONENT ADUSTEMENT
1CFE'                   FDGOON3:

1CFE'   DD 70 00                LD      (IX+FPWS+FEXP1-MEMBEG),B        ;NEW EXPONENT
1D01'   D1                      POP     DE
1D02'   C3 1C04'                JP      FCORR           ;RESULT AJUSTEMENT
                        ;================================================================
1D05'   46 4E 45 47             DB      'FNEGAT','E' OR CLAST
1D09'   41 54 C5
1D0C'   1C7A'                   DW      FDIV-1
1D0E'   07                      DB      7
1D0F'                   FNEGATE:
1D0F'   1D11'                   DW      $+2

                                RSTPULL
1D11'   DF              +       RST     018H
1D12'   7A                      LD      A,D
1D13'   A7                      AND     A
1D14'   28 02                   JR      Z,FNQUIT
1D16'   EE 80                   XOR     80H             ;NEGATE NUMBERS <> 0
1D18'                   FNQUIT:
1D18'   57                      LD      D,A
                                RSTPUSH
1D19'   D7              +       RST     010H
1D1A'   FD E9                   JP      (IY)
                        ;================================================================
1D1C'   49 4E D4                DB      'IN','T' OR CLAST
1D1F'   1D0E'                   DW      FNEGATE-1
1D21'   03                      DB      3
1D22'                   INT:
1D22'   1D24'                   DW      $+2

1D24'   2A 3C3B                 LD      HL,(SPARE)
1D27'   2B                      DEC     HL
1D28'   11 0000                 LD      DE,0            ;CLEAR VALUE
1D2B'                   INTLOOP:
1D2B'   7E                      LD      A,(HL)          ;GET EXPONENT
1D2C'   07                      RLCA
1D2D'   FE 82                   CP      0+(FEOFFS+1) SHL 1
1D2F'   38 14                   JR      C,INTQUIT       ;ABS (NUMBER) <1.0?

1D31'   AF                      XOR     A
1D32'   2B                      DEC     HL
1D33'   CD 0732'                CALL    DECSTORE        ;SHIFT LEFT BY A DIGIT
1D36'   23                      INC     HL
1D37'   EB                      EX      DE,HL
1D38'   44                      LD      B,H
1D39'   4D                      LD      C,L
1D3A'   29                      ADD     HL,HL
1D3B'   29                      ADD     HL,HL
1D3C'   09                      ADD     HL,BC
1D3D'   29                      ADD     HL,HL           ;VALUE * 10
1D3E'   4F                      LD      C,A
1D3F'   06 00                   LD      B,0
1D41'   09                      ADD     HL,BC           ;ADD OVERFLOW DIGIT
1D42'   EB                      EX      DE,HL
```

```
1D43'   18 E6                           JR      INTLOOP

1D45'                           INTQUIT:
1D45'   2B                              DEC     HL
1D46'   2B                              DEC     HL
1D47'   72                              LD      (HL),D
1D48'   2B                              DEC     HL
1D49'   73                              LD      (HL),E
1D4A'   11 0D94'                        LD      DE,IFN0NEG
1D4D'   C3 04BF'                        JP      NEXTDE          ;ADJUST THE SIGN
                                ;=================================================================
1D50'   55 46 4C 4F                     DB      'UFLOA','T' OR CLAST
1D54'   41 D4
1D56'   1D21'                           DW      INT-1
1D58'   06                              DB      6
1D59'                           UFLOAT:
1D59'   1D5B'                           DW      $+2

                                        RSTPULL
1D5B'   DF              +               RST     018H
1D5C'   EB                              EX      DE,HL
1D5D'   01 1000                         LD      BC,0 OR (16 SHL 8)
1D60'   51                              LD      D,C
1D61'   59                              LD      E,C
1D62'                           UFLOOP:
1D62'   29                              ADD     HL,HL
1D63'   7B                              LD      A,E
1D64'   8F                              ADC     A,A
1D65'   27                              DAA
1D66'   5F                              LD      E,A
1D67'   7A                              LD      A,D
1D68'   8F                              ADC     A,A
1D69'   27                              DAA
1D6A'   57                              LD      D,A
1D6B'   CB 11                           RL      C
1D6D'   10 F3                           DJNZ    UFLOOP          ;CONVERT TO BCD NUMBER

                                        RSTPUSH
1D6F'   D7              +               RST     010H
1D70'   16 46                           LD      D,FEOFFS+6
1D72'   59                              LD      E,C
                                        RSTPUSH                 ;SAVE NUMBER
1D73'   D7              +               RST     010H
1D74'   2B                              DEC     HL
1D75'   2B                              DEC     HL
1D76'   CD 0740'                        CALL    FZEROEQ         ;ADJUST EXPONENT ON 0
1D79'   FD E9                           JP      (IY)
                                ;=================================================================
                                ;       CHARACTER SET
1D7B'   00 00 00 00                     DB      000H,000H,000H,000H
1D7F'   00 00 00                        DB      000H,000H,000H          ;........
                                                                        ;........
                                                                        ;........
                                                                        ;........
                                                                        ;........
                                                                        ;........
```

```
                                                               ;........
1D82'   10 10 10 10                 DB      010H,010H,010H,010H
1D86'   00 10 00                    DB      000H,010H,000H     ;...*....
                                                               ;...*....
                                                               ;...*....
                                                               ;...*....
                                                               ;........
                                                               ;...*....
                                                               ;........
1D89'   24 24 00 00                 DB      024H,024H,000H,000H
1D8D'   00 00 00                    DB      000H,000H,000H     ;..*..*..
                                                               ;..*..*..
                                                               ;........
                                                               ;........
                                                               ;........
                                                               ;........
                                                               ;........
1D90'   24 7E 24 24                 DB      024H,07EH,024H,024H
1D94'   7E 24 00                    DB      07EH,024H,000H     ;..*..*..
                                                               ;.******.
                                                               ;..*..*..
                                                               ;..*..*..
                                                               ;.******.
                                                               ;..*..*..
                                                               ;........
1D97'   08 3E 28 3E                 DB      008H,03EH,028H,03EH
1D9B'   0A 3E 08                    DB      00AH,03EH,008H     ;....*...
                                                               ;..*****.
                                                               ;..*.*...
                                                               ;..*****.
                                                               ;....*.*.
                                                               ;..*****.
                                                               ;....*...
1D9E'   62 64 08 10                 DB      062H,064H,008H,010H
1DA2'   26 46 00                    DB      026H,046H,000H     ;.**...*.
                                                               ;.**..*..
                                                               ;....*...
                                                               ;...*....
                                                               ;..*..**.
                                                               ;.*...**.
                                                               ;........
1DA5'   10 28 10 2A                 DB      010H,028H,010H,02AH
1DA9'   44 3A 00                    DB      044H,03AH,000H     ;...*....
                                                               ;..*.*...
                                                               ;...*....
                                                               ;..*.*.*.
                                                               ;.*...*..
                                                               ;..***.*.
                                                               ;........
1DAC'   08 10 00 00                 DB      008H,010H,000H,000H
1DB0'   00 00 00                    DB      000H,000H,000H     ;....*...
                                                               ;...*....
                                                               ;........
                                                               ;........
                                                               ;........
                                                               ;........
```

```
                                                                    ;........
1DB3'   04 08 08 08                 DB      004H,008H,008H,008H
1DB7'   08 04 00                    DB      008H,004H,000H          ;.....*..
                                                                    ;....*...
                                                                    ;....*...
                                                                    ;....*...
                                                                    ;....*...
                                                                    ;.....*..
                                                                    ;........
1DBA'   20 10 10 10                 DB      020H,010H,010H,010H
1DBE'   10 20 00                    DB      010H,020H,000H          ;..*.....
                                                                    ;...*....
                                                                    ;...*....
                                                                    ;...*....
                                                                    ;...*....
                                                                    ;..*.....
                                                                    ;........
1DC1'   00 14 08 3E                 DB      000H,014H,008H,03EH
1DC5'   08 14 00                    DB      008H,014H,000H          ;........
                                                                    ;...*.*..
                                                                    ;....*...
                                                                    ;..*****.
                                                                    ;....*...
                                                                    ;...*.*..
                                                                    ;........
1DC8'   00 08 08 3E                 DB      000H,008H,008H,03EH
1DCC'   08 08 00                    DB      008H,008H,000H          ;........
                                                                    ;....*...
                                                                    ;....*...
                                                                    ;..*****.
                                                                    ;....*...
                                                                    ;....*...
                                                                    ;........
1DCF'   00 00 00 00                 DB      000H,000H,000H,000H
1DD3'   08 08 10                    DB      008H,008H,010H          ;........
                                                                    ;........
                                                                    ;........
                                                                    ;........
                                                                    ;....*...
                                                                    ;....*...
                                                                    ;...*....
1DD6'   00 00 00 3E                 DB      000H,000H,000H,03EH
1DDA'   00 00 00                    DB      000H,000H,000H          ;........
                                                                    ;........
                                                                    ;........
                                                                    ;..*****.
                                                                    ;........
                                                                    ;........
                                                                    ;........
1DDD'   00 00 00 00                 DB      000H,000H,000H,000H
1DE1'   18 18 00                    DB      018H,018H,000H          ;........
                                                                    ;........
                                                                    ;........
                                                                    ;........
                                                                    ;...**...
                                                                    ;...**...
```

```
                                                        ;........
1DE4'   00 02 04 08             DB      000H,002H,004H,008H
1DE8'   10 20 00                DB      010H,020H,000H  ;........
                                                        ;......*.
                                                        ;.....*..
                                                        ;....*...
                                                        ;...*....
                                                        ;..*.....
                                                        ;........
1DEB'   3C 46 4A 52             DB      03CH,046H,04AH,052H
1DEF'   62 3C 00                DB      062H,03CH,000H  ;..****..
                                                        ;.*...**.
                                                        ;.*..*.*.
                                                        ;.*.*..*.
                                                        ;.**...*.
                                                        ;..****..
                                                        ;........
1DF2'   18 28 08 08             DB      018H,028H,008H,008H
1DF6'   08 3E 00                DB      008H,03EH,000H  ;...**...
                                                        ;..*.*...
                                                        ;....*...
                                                        ;....*...
                                                        ;....*...
                                                        ;..*****.
                                                        ;........
1DF9'   3C 42 02 3C             DB      03CH,042H,002H,03CH
1DFD'   40 7E 00                DB      040H,07EH,000H  ;..****..
                                                        ;.*....*.
                                                        ;......*.
                                                        ;..****..
                                                        ;.*......
                                                        ;.******.
                                                        ;........
1E00'   3C 42 0C 02             DB      03CH,042H,00CH,002H
1E04'   42 3C 00                DB      042H,03CH,000H  ;..****..
                                                        ;.*....*.
                                                        ;....**..
                                                        ;......*.
                                                        ;.*....*.
                                                        ;..****..
                                                        ;........
1E07'   08 18 28 48             DB      008H,018H,028H,048H
1E0B'   7E 08 00                DB      07EH,008H,000H  ;....*...
                                                        ;...**...
                                                        ;..*.*...
                                                        ;.*..*...
                                                        ;.******.
                                                        ;....*...
                                                        ;........
1E0E'   7E 40 7C 02             DB      07EH,040H,07CH,002H
1E12'   42 3C 00                DB      042H,03CH,000H  ;.******.
                                                        ;.*......
                                                        ;.*****..
                                                        ;......*.
                                                        ;.*....*.
                                                        ;..****..
```

```
                                                        ;........
1E15'   3C 40 7C 42                 DB      03CH,040H,07CH,042H
1E19'   42 3C 00                    DB      042H,03CH,000H      ;..****..
                                                        ;.*......
                                                        ;.*****..
                                                        ;.*....*.
                                                        ;.*....*.
                                                        ;..****..
                                                        ;........
1E1C'   7E 02 04 08                 DB      07EH,002H,004H,008H
1E20'   10 10 00                    DB      010H,010H,000H      ;.******.
                                                        ;......*.
                                                        ;.....*..
                                                        ;....*...
                                                        ;...*....
                                                        ;...*....
                                                        ;........
1E23'   3C 42 3C 42                 DB      03CH,042H,03CH,042H
1E27'   42 3C 00                    DB      042H,03CH,000H      ;..****..
                                                        ;.*....*.
                                                        ;..****..
                                                        ;.*....*.
                                                        ;.*....*.
                                                        ;..****..
                                                        ;........
1E2A'   3C 42 42 3E                 DB      03CH,042H,042H,03EH
1E2E'   02 3C 00                    DB      002H,03CH,000H      ;..****..
                                                        ;.*....*.
                                                        ;.*....*.
                                                        ;..*****.
                                                        ;......*.
                                                        ;..****..
                                                        ;........
1E31'   00 00 10 00                 DB      000H,000H,010H,000H
1E35'   00 10 00                    DB      000H,010H,000H      ;........
                                                        ;........
                                                        ;...*....
                                                        ;........
                                                        ;........
                                                        ;...*....
                                                        ;........
1E38'   00 10 00 00                 DB      000H,010H,000H,000H
1E3C'   10 10 20                    DB      010H,010H,020H      ;........
                                                        ;...*....
                                                        ;........
                                                        ;........
                                                        ;...*....
                                                        ;...*....
                                                        ;..*.....
1E3F'   00 04 08 10                 DB      000H,004H,008H,010H
1E43'   08 04 00                    DB      008H,004H,000H      ;........
                                                        ;.....*..
                                                        ;....*...
                                                        ;...*....
                                                        ;....*...
                                                        ;.....*..
```

```
                                                                ;........
1E46'   00 00 3E 00                 DB      000H,000H,03EH,000H
1E4A'   3E 00 00                    DB      03EH,000H,000H      ;........
                                                                ;........
                                                                ;..*****.
                                                                ;........
                                                                ;..*****.
                                                                ;........
                                                                ;........
1E4D'   00 10 08 04                 DB      000H,010H,008H,004H
1E51'   08 10 00                    DB      008H,010H,000H      ;........
                                                                ;...*....
                                                                ;....*...
                                                                ;.....*..
                                                                ;....*...
                                                                ;...*....
                                                                ;........
1E54'   3C 42 04 08                 DB      03CH,042H,004H,008H
1E58'   00 08                       DB      000H,008H           ;..****..
                                                                ;.*....*.
                                                                ;.....*..
                                                                ;....*...
                                                                ;........
                                                                ;....*...
1E5A'   3C 4A 56 5E                 DB      03CH,04AH,056H,05EH
1E5E'   40 3C                       DB      040H,03CH           ;..****..
                                                                ;.*..*.*.
                                                                ;.*.*.**.
                                                                ;.*.****.
                                                                ;.*......
                                                                ;..****..
1E60'   3C 42 42 7E                 DB      03CH,042H,042H,07EH
1E64'   42 42                       DB      042H,042H           ;..****..
                                                                ;.*....*.
                                                                ;.*....*.
                                                                ;.******.
                                                                ;.*....*.
                                                                ;.*....*.
1E66'   7C 42 7C 42                 DB      07CH,042H,07CH,042H
1E6A'   42 7C                       DB      042H,07CH           ;.*****..
                                                                ;.*....*.
                                                                ;.*****..
                                                                ;.*....*.
                                                                ;.*....*.
                                                                ;.*****..
1E6C'   3C 42 40 40                 DB      03CH,042H,040H,040H
1E70'   42 3C                       DB      042H,03CH           ;..****..
                                                                ;.*....*.
                                                                ;.*......
                                                                ;.*......
                                                                ;.*....*.
                                                                ;..****..
1E72'   78 44 42 42                 DB      078H,044H,042H,042H
1E76'   44 78                       DB      044H,078H           ;.****...
                                                                ;.*...*..
                                                                ;.*....*.
```

```
                                                        ;.*....*.
                                                        ;.*...*..
                                                        ;.****...
1E78'   7E 40 7C 40             DB      07EH,040H,07CH,040H
1E7C'   40 7E                   DB      040H,07EH       ;.******.
                                                        ;.*......
                                                        ;.*****..
                                                        ;.*......
                                                        ;.*......
                                                        ;.******.
1E7E'   7E 40 7C 40             DB      07EH,040H,07CH,040H
1E82'   40 40                   DB      040H,040H       ;.******.
                                                        ;.*......
                                                        ;.*****..
                                                        ;.*......
                                                        ;.*......
                                                        ;.*......
1E84'   3C 42 40 4E             DB      03CH,042H,040H,04EH
1E88'   42 3C                   DB      042H,03CH       ;..****..
                                                        ;.*....*.
                                                        ;.*......
                                                        ;.*..***.
                                                        ;.*....*.
                                                        ;..****..
1E8A'   42 42 7E 42             DB      042H,042H,07EH,042H
1E8E'   42 42                   DB      042H,042H       ;.*....*.
                                                        ;.*....*.
                                                        ;.******.
                                                        ;.*....*.
                                                        ;.*....*.
                                                        ;.*....*.
1E90'   3E 08 08 08             DB      03EH,008H,008H,008H
1E94'   08 3E                   DB      008H,03EH       ;..*****.
                                                        ;....*...
                                                        ;....*...
                                                        ;....*...
                                                        ;....*...
                                                        ;..*****.
1E96'   02 02 02 42             DB      002H,002H,002H,042H
1E9A'   42 3C                   DB      042H,03CH       ;......*.
                                                        ;......*.
                                                        ;......*.
                                                        ;.*....*.
                                                        ;.*....*.
                                                        ;..****..
1E9C'   44 48 70 48             DB      044H,048H,070H,048H
1EA0'   44 42                   DB      044H,042H       ;.*...*..
                                                        ;.*..*...
                                                        ;.***....
                                                        ;.*..*...
                                                        ;.*...*..
                                                        ;.*....*.
1EA2'   40 40 40 40             DB      040H,040H,040H,040H
1EA6'   40 7E                   DB      040H,07EH       ;.*......
                                                        ;.*......
                                                        ;.*......
```

```
                                                        ;.*......
                                                        ;.*......
                                                        ;.******.
1EA8'   42 66 5A 42         DB      042H,066H,05AH,042H
1EAC'   42 42               DB      042H,042H           ;.*....*.
                                                        ;.**..**.
                                                        ;.*.**.*.
                                                        ;.*....*.
                                                        ;.*....*.
                                                        ;.*....*.
1EAE'   42 62 52 4A         DB      042H,062H,052H,04AH
1EB2'   46 42               DB      046H,042H           ;.*....*.
                                                        ;.**...*.
                                                        ;.*.*..*.
                                                        ;.*..*.*.
                                                        ;.*...**.
                                                        ;.*....*.
1EB4'   3C 42 42 42         DB      03CH,042H,042H,042H
1EB8'   42 3C               DB      042H,03CH           ;..****..
                                                        ;.*....*.
                                                        ;.*....*.
                                                        ;.*....*.
                                                        ;.*....*.
                                                        ;..****..
1EBA'   7C 42 42 7C         DB      07CH,042H,042H,07CH
1EBE'   40 40               DB      040H,040H           ;.*****..
                                                        ;.*....*.
                                                        ;.*....*.
                                                        ;.*****..
                                                        ;.*......
                                                        ;.*......
1EC0'   3C 42 42 52         DB      03CH,042H,042H,052H
1EC4'   4A 3C               DB      04AH,03CH           ;..****..
                                                        ;.*....*.
                                                        ;.*....*.
                                                        ;.*.*..*.
                                                        ;.*..*.*.
                                                        ;..****..
1EC6'   7C 42 42 7C         DB      07CH,042H,042H,07CH
1ECA'   44 42               DB      044H,042H           ;.*****..
                                                        ;.*....*.
                                                        ;.*....*.
                                                        ;.*****..
                                                        ;.*...*..
                                                        ;.*....*.
1ECC'   3C 40 3C 02         DB      03CH,040H,03CH,002H
1ED0'   42 3C               DB      042H,03CH           ;..****..
                                                        ;.*......
                                                        ;..****..
                                                        ;......*.
                                                        ;.*....*.
                                                        ;..****..
1ED2'   FE 10 10 10         DB      0FEH,010H,010H,010H
1ED6'   10 10               DB      010H,010H           ;*******.
                                                        ;...*....
                                                        ;...*....
```

```
                                                        ;...*....
                                                        ;...*....
                                                        ;...*....
1ED8'   42 42 42 42         DB      042H,042H,042H,042H
1EDC'   42 3E               DB      042H,03EH           ;.*....*.
                                                        ;.*....*.
                                                        ;.*....*.
                                                        ;.*....*.
                                                        ;.*....*.
                                                        ;..*****.
1EDE'   42 42 42 42         DB      042H,042H,042H,042H
1EE2'   24 18               DB      024H,018H           ;.*....*.
                                                        ;.*....*.
                                                        ;.*....*.
                                                        ;.*....*.
                                                        ;..*..*..
                                                        ;...**...
1EE4'   42 42 42 42         DB      042H,042H,042H,042H
1EE8'   5A 24               DB      05AH,024H           ;.*....*.
                                                        ;.*....*.
                                                        ;.*....*.
                                                        ;.*....*.
                                                        ;.*.**.*.
                                                        ;..*..*..
1EEA'   42 24 18 18         DB      042H,024H,018H,018H
1EEE'   24 42               DB      024H,042H           ;.*....*.
                                                        ;..*..*..
                                                        ;...**...
                                                        ;...**...
                                                        ;..*..*..
                                                        ;.*....*.
1EF0'   82 44 28 10         DB      082H,044H,028H,010H
1EF4'   10 10               DB      010H,010H           ;*.....*.
                                                        ;.*...*..
                                                        ;..*.*...
                                                        ;...*....
                                                        ;...*....
                                                        ;...*....
1EF6'   7E 04 08 10         DB      07EH,004H,008H,010H
1EFA'   20 7E               DB      020H,07EH           ;.******.
                                                        ;.....*..
                                                        ;....*...
                                                        ;...*....
                                                        ;..*.....
                                                        ;.******.
1EFC'   0E 08 08 08         DB      00EH,008H,008H,008H
1F00'   08 0E               DB      008H,00EH           ;....***.
                                                        ;....*...
                                                        ;....*...
                                                        ;....*...
                                                        ;....*...
                                                        ;....***.
1F02'   00 40 20 10         DB      000H,040H,020H,010H
1F06'   08 04               DB      008H,004H           ;........
                                                        ;.*......
                                                        ;..*.....
```

```
                                                        ;...*....
                                                        ;....*...
                                                        ;.....*..
1F08'   70 10 10 10          DB      070H,010H,010H,010H
1F0C'   10 70                DB      010H,070H          ;.***....
                                                        ;...*....
                                                        ;...*....
                                                        ;...*....
                                                        ;...*....
                                                        ;.***....
1F0E'   10 38 54 10          DB      010H,038H,054H,010H
1F12'   10 10                DB      010H,010H          ;...*....
                                                        ;..***...
                                                        ;.*.*.*..
                                                        ;...*....
                                                        ;...*....
                                                        ;...*....
1F14'   00 00 00 00          DB      000H,000H,000H,000H
1F18'   00 00 FF             DB      000H,000H,0FFH     ;........
                                                        ;........
                                                        ;........
                                                        ;........
                                                        ;........
                                                        ;........
                                                        ;********
1F1B'   1C 22 78 20          DB      01CH,022H,078H,020H
1F1F'   20 7E 00             DB      020H,07EH,000H     ;...***..
                                                        ;..*...*.
                                                        ;.****...
                                                        ;..*.....
                                                        ;..*.....
                                                        ;.******.
                                                        ;........
1F22'   00 38 04 3C          DB      000H,038H,004H,03CH
1F26'   44 3E 00             DB      044H,03EH,000H     ;........
                                                        ;..***...
                                                        ;.....*..
                                                        ;..****..
                                                        ;.*...*..
                                                        ;..*****.
                                                        ;........
1F29'   20 20 3C 22          DB      020H,020H,03CH,022H
1F2D'   22 3C 00             DB      022H,03CH,000H     ;..*.....
                                                        ;..*.....
                                                        ;..****..
                                                        ;..*...*.
                                                        ;..*...*.
                                                        ;..****..
                                                        ;........
1F30'   00 1C 20 20          DB      000H,01CH,020H,020H
1F34'   20 1C 00             DB      020H,01CH,000H     ;........
                                                        ;...***..
                                                        ;..*.....
                                                        ;..*.....
                                                        ;..*.....
                                                        ;...***..
```

```
                                                                ;........
1F37'   04 04 3C 44                 DB      004H,004H,03CH,044H
1F3B'   44 3E 00                    DB      044H,03EH,000H      ;.....*..
                                                                ;.....*..
                                                                ;..****..
                                                                ;.*...*..
                                                                ;.*...*..
                                                                ;..*****.
                                                                ;........
1F3E'   00 38 44 78                 DB      000H,038H,044H,078H
1F42'   40 3C 00                    DB      040H,03CH,000H      ;........
                                                                ;..***...
                                                                ;.*...*..
                                                                ;.****...
                                                                ;.*......
                                                                ;..****..
                                                                ;........
1F45'   0C 10 18 10                 DB      00CH,010H,018H,010H
1F49'   10 10 00                    DB      010H,010H,000H      ;....**..
                                                                ;...*....
                                                                ;...**...
                                                                ;...*....
                                                                ;...*....
                                                                ;...*....
                                                                ;........
1F4C'   00 3C 44 44                 DB      000H,03CH,044H,044H
1F50'   3C 04 38                    DB      03CH,004H,038H      ;........
                                                                ;..****..
                                                                ;.*...*..
                                                                ;.*...*..
                                                                ;..****..
                                                                ;.....*..
                                                                ;..***...
1F53'   40 40 78 44                 DB      040H,040H,078H,044H
1F57'   44 44 00                    DB      044H,044H,000H      ;.*......
                                                                ;.*......
                                                                ;.****...
                                                                ;.*...*..
                                                                ;.*...*..
                                                                ;.*...*..
                                                                ;........
1F5A'   10 00 30 10                 DB      010H,000H,030H,010H
1F5E'   10 38 00                    DB      010H,038H,000H      ;...*....
                                                                ;........
                                                                ;..**....
                                                                ;...*....
                                                                ;...*....
                                                                ;..***...
                                                                ;........
1F61'   04 00 04 04                 DB      004H,000H,004H,004H
1F65'   04 24 18                    DB      004H,024H,018H      ;.....*..
                                                                ;........
                                                                ;.....*..
                                                                ;.....*..
                                                                ;.....*..
                                                                ;..*..*..
```

```
                                                                        ;...**...
1F68'   20 28 30 30                     DB      020H,028H,030H,030H
1F6C'   28 24 00                        DB      028H,024H,000H          ;..*.....
                                                                        ;..*.*...
                                                                        ;..**....
                                                                        ;..**....
                                                                        ;..*.*...
                                                                        ;..*..*..
                                                                        ;........
1F6F'   10 10 10 10                     DB      010H,010H,010H,010H
1F73'   10 0C 00                        DB      010H,00CH,000H          ;...*....
                                                                        ;...*....
                                                                        ;...*....
                                                                        ;...*....
                                                                        ;...*....
                                                                        ;....**..
                                                                        ;........
1F76'   00 68 54 54                     DB      000H,068H,054H,054H
1F7A'   54 54 00                        DB      054H,054H,000H          ;........
                                                                        ;.**.*...
                                                                        ;.*.*.*..
                                                                        ;.*.*.*..
                                                                        ;.*.*.*..
                                                                        ;.*.*.*..
                                                                        ;........
1F7D'   00 78 44 44                     DB      000H,078H,044H,044H
1F81'   44 44 00                        DB      044H,044H,000H          ;........
                                                                        ;.****...
                                                                        ;.*...*..
                                                                        ;.*...*..
                                                                        ;.*...*..
                                                                        ;.*...*..
                                                                        ;........
1F84'   00 38 44 44                     DB      000H,038H,044H,044H
1F88'   44 38 00                        DB      044H,038H,000H          ;........
                                                                        ;..***...
                                                                        ;.*...*..
                                                                        ;.*...*..
                                                                        ;.*...*..
                                                                        ;..***...
                                                                        ;........
1F8B'   00 78 44 44                     DB      000H,078H,044H,044H
1F8F'   78 40 40                        DB      078H,040H,040H          ;........
                                                                        ;.****...
                                                                        ;.*...*..
                                                                        ;.*...*..
                                                                        ;.****...
                                                                        ;.*......
                                                                        ;.*......
1F92'   00 3C 44 44                     DB      000H,03CH,044H,044H
1F96'   3C 04 06                        DB      03CH,004H,006H          ;........
                                                                        ;..****..
                                                                        ;.*...*..
                                                                        ;.*...*..
                                                                        ;..****..
                                                                        ;.....*..
```

```
                                                                ;.....**.
1F99'   00 1C 20 20                 DB      000H,01CH,020H,020H
1F9D'   20 20 00                    DB      020H,020H,000H      ;........
                                                                ;...***..
                                                                ;..*.....
                                                                ;..*.....
                                                                ;..*.....
                                                                ;..*.....
                                                                ;........
1FA0'   00 38 40 38                 DB      000H,038H,040H,038H
1FA4'   04 78 00                    DB      004H,078H,000H      ;........
                                                                ;..***...
                                                                ;.*......
                                                                ;..***...
                                                                ;.....*..
                                                                ;.****...
                                                                ;........
1FA7'   10 38 10 10                 DB      010H,038H,010H,010H
1FAB'   10 0C 00                    DB      010H,00CH,000H      ;...*....
                                                                ;..***...
                                                                ;...*....
                                                                ;...*....
                                                                ;...*....
                                                                ;....**..
                                                                ;........
1FAE'   00 44 44 44                 DB      000H,044H,044H,044H
1FB2'   44 3C 00                    DB      044H,03CH,000H      ;........
                                                                ;.*...*..
                                                                ;.*...*..
                                                                ;.*...*..
                                                                ;.*...*..
                                                                ;..****..
                                                                ;........
1FB5'   00 44 44 28                 DB      000H,044H,044H,028H
1FB9'   28 10 00                    DB      028H,010H,000H      ;........
                                                                ;.*...*..
                                                                ;.*...*..
                                                                ;..*.*...
                                                                ;..*.*...
                                                                ;...*....
                                                                ;........
1FBC'   00 44 54 54                 DB      000H,044H,054H,054H
1FC0'   54 28 00                    DB      054H,028H,000H      ;........
                                                                ;.*...*..
                                                                ;.*.*.*..
                                                                ;.*.*.*..
                                                                ;.*.*.*..
                                                                ;..*.*...
                                                                ;........
1FC3'   00 44 28 10                 DB      000H,044H,028H,010H
1FC7'   28 44 00                    DB      028H,044H,000H      ;........
                                                                ;.*...*..
                                                                ;..*.*...
                                                                ;...*....
                                                                ;..*.*...
                                                                ;.*...*..
```

```
                                                            ;........
1FCA'   00 44 44 44                DB      000H,044H,044H,044H
1FCE'   3C 04 38                   DB      03CH,004H,038H       ;........
                                                            ;.*...*..
                                                            ;.*...*..
                                                            ;.*...*..
                                                            ;..****..
                                                            ;.....*..
                                                            ;..***...
1FD1'   00 7C 08 10                DB      000H,07CH,008H,010H
1FD5'   20 7C 00                   DB      020H,07CH,000H       ;........
                                                            ;.*****..
                                                            ;....*...
                                                            ;...*....
                                                            ;..*.....
                                                            ;.*****..
                                                            ;........
1FD8'   0E 08 30 30                DB      00EH,008H,030H,030H
1FDC'   08 0E 00                   DB      008H,00EH,000H       ;....***.
                                                            ;....*...
                                                            ;..**....
                                                            ;..**....
                                                            ;....*...
                                                            ;....***.
                                                            ;........
1FDF'   08 08 08 08                DB      008H,008H,008H,008H
1FE3'   08 08 00                   DB      008H,008H,000H       ;....*...
                                                            ;....*...
                                                            ;....*...
                                                            ;....*...
                                                            ;....*...
                                                            ;....*...
                                                            ;........
1FE6'   70 10 0C 0C                DB      070H,010H,00CH,00CH
1FEA'   10 70 00                   DB      010H,070H,000H       ;.***....
                                                            ;...*....
                                                            ;....**..
                                                            ;....**..
                                                            ;...*....
                                                            ;.***....
                                                            ;........
1FED'   32 4C 00 00                DB      032H,04CH,000H,000H
1FF1'   00 00 00                   DB      000H,000H,000H       ;..**..*.
                                                            ;.*..**..
                                                            ;........
                                                            ;........
                                                            ;........
                                                            ;........
                                                            ;........
1FF4'   3C 42 99 A1                DB      03CH,042H,099H,0A1H
1FF8'   A1 99 42 3C                DB      0A1H,099H,042H,03CH  ;..****..
                                                            ;.*....*.
                                                            ;*..**..*
                                                            ;*.*....*
                                                            ;*.*....*
                                                            ;*..**..*
```

```
                                                                        ;.*....*.
                                                                        ;..****..
1FFC'                           ROMCHR:
                                ;====================================================================
1FFC'   FF                              DB      0FFH
1FFD'   1D58'                           DW      UFLOAT-1
1FFF'   00                              DB      000H
                                ;====================================================================
                                        END
```

Macros:
RSTEMIT RSTERR RSTPULL RSTPUSH

Symbols:

00DE'	ABGOON	00AB'	ABORT	00FF'	ABORTEND		
0C0D'	ABS	0F9E'	ALLOC	0F76'	ALLOT		
0F83'	ALLOT2	1028'	ASCII	12D8'	ASSERT		
08B3'	AT	0B19'	ATPOS	048A'	BASE		
0BC7'	BDBREAK	0BCB'	BDLOOP	0B98'	BEEP		
0BC9'	BEEPDELAY	121A'	BEGIN	07FA'	BLANKS		
07FB'	BLANKS2	07FE'	BLLOOP	1954'	BLOAD		
0BAF'	BLOOP	07DA'	BLWORD	04F0'	BREAK		
1944'	BSAVE	1979'	BVERIFY	1592'	CADICT		
10A7'	CALL	0896'	CAT	0B28'	CATPOS		
1584'	CAWORD	0A24'	CCLS	000D	CCR		
153A'	CDCOLON	1537'	CDDEFCOM	0005	CDL		
150B'	CDLOOP	153F'	CDSETCTXT	08A5'	CEXCLAM		
0F09'	CHGOON	054F'	CHKIMM	0561'	CHKIQUIT		
0564'	CHKNUMBER	061B'	CHKSTRING	0F1D'	CHLOOP		
2C00	CHRSET	0080	CINV	0F5F'	CKOMMA		
0080	CLAST	0A1D'	CLS	0C21'	CMPPUSH		
07B8'	CNVDIGIT	07CD'	CNVDOK	07D7'	CNVDQUIT		
077B'	CNVEND	074C'	CNVINT	07B4'	CNVTEND		
078C'	CNVTLOOP	0EAF'	COLON	10F5'	COMPILER		
0FE2'	CONSTANT	0473'	CONTEXT	078A'	CONVERT		
1568'	CORRADDR	14F8'	CORRCURR	1504'	CORRDICT		
1557'	CORRPTR	1548'	CORRWORD	094D'	CPICK		
095B'	CPKGOON	007F	CPR	0018'	CPULL		
0010'	CPUSH	0A95'	CR	0ED0'	CREATE		
0EFB'	CRHEADER	097F'	CTYPE	0480'	CURRENT		
3C20	CURSOR	05FC'	CWEND	0614'	CWERR		
05EA'	CWLOOP1	05F3'	CWLOOP2	0600'	CWNFND		
05E1'	CWORD	0CD5'	D32GOON	0CDB'	D32LOOP		
0CE5'	D32NEXT	1ADF'	DBRGOON1	1AE9'	DBRGOON2		
1AC8'	DBVLOOP	023F'	DCCDGOON	022C'	DCCHARDEL		
02D8'	DCCLEAR	0225'	DCCURDEL	01CE'	DCDCEND		
0198'	DCDCINS	0196'	DCDCNORM	01E4'	DCDCQUIT		
01A6'	DCDCSCROL	01C9'	DCDCSLOOP	01DD'	DCDCSTORE		
029C'	DCDNLOOP	017E'	DCDOCHAR	0295'	DCDOWN		
02A2'	DCDSCROLL	02D0'	DCENTER	01FE'	DCFLAG		
0302'	DCGETCIN	01F0'	DCJMPTAB	02CA'	DCLDLOOP		
0204'	DCLEFT	02C3'	DCLINEDEL	0210'	DCNOP		
0276'	DCOUTCUR	02EA'	DCRETYPE	0211'	DCRIGHT		
02F9'	DCSELOOP	02ED'	DCSETBEG	0282'	DCSETCUR		
02F4'	DCSETEND	02B0'	DCSTREND	0247'	DCUP		
024E'	DCUPLOOP	0254'	DCUSCROLL	0269'	DCUSLOOP		
0723'	DECGET	0EA3'	DECIMAL	0443'	DECLINE		
072C'	DECSHIN	0732'	DECSTORE	1074'	DEFINER		
11AB'	DEFINITIONS	14DC'	DELWORD	3C39	DICT		
3C40	DICT1ST	14DA'	DICTERR	0D51'	DIV		
0CC4'	DIV32BY16	0D00'	DIVMOD	0D0D'	DIVMOD2		
044B'	DLEND	0C83'	DLT	0DBA'	DNEGATE		
0DC5'	DNLOOP	12AB'	DO	129F'	DOBEGIN		
1AB8'	DOBLOAD	1AD0'	DOBREAD	1ABE'	DOBVERIFY		
0EC3'	DOCOL	1110'	DOCOMGOON	1108'	DOCOMPILER		
0FF5'	DOCONSTANT	0FEC'	DOCREATE	01E6'	DOCTRL		

1085'	DODEFINER	1323'	DODO	10E8'	DODOESGT
1271'	DOELSE	10B4'	DOESGT	10CD'	DOESPATCH
1225'	DOFPATCH	1283'	DOIF	1379'	DOLBRACKET
1332'	DOLOOP	0DBF'	DONEGATE	133C'	DOPLUSLOOP
1396'	DOPTSTR	1276'	DOREPEAT	1237'	DORPATCH
1140'	DORUNSGT	1A4F'	DOSAVE	12A4'	DOTHEN
128D'	DOUNTIL	0FF0'	DOVARIABLE	197D'	DOVERIFY
1288'	DOWHILE	0490'	DP	0DEE'	DPLUS
0879'	DROP	109A'	DROPGOON	0736'	DSLOOP
086B'	DUP	165E'	EDIT	1675'	EDITLIST
16DF'	ELACK	1697'	ELCOLON	169C'	ELCOMPILER
16B1'	ELDEFINER	16E8'	ELEDIT	16C7'	ELLLOOP
16C3'	ELMLOOP	16B4'	ELOUT	1702'	ELQUIT
16D2'	ELREADY	11EC'	ELSE	0AA3'	EMIT
03FF'	EMITSCR	3C22	ENDBUF	0C4A'	EQ
1294'	EQUJUMP	0009	ERRAT	0005	ERRBLK
0003	ERRBRK	000B	ERRDICT	000D	ERRFIND
0008	ERRFLT	0004	ERRIMM	000E	ERRLIST
0001	ERRMEM	000C	ERRMODE	0006	ERRNAME
3C3D	ERRNO	FFFF	ERRNONE	04D7'	ERRORSTK
0007	ERRPICK	000A	ERRREAD	0002	ERRSTK
0416'	ESENTER	041C'	ESQUIT	08C1'	EXCLAM
1815'	EXECDE	069A'	EXECUTE	13F0'	EXIT
3C29	EXWRCH	0003	FACCU	1B53'	FADDITION
1BB3'	FADDSUB	1B28'	FADJLP1	1B2E'	FADJLP2
1B3A'	FADJLP3	1B22'	FADJUST	1BC1'	FASGOON1
1BC9'	FASGOON2	1C02'	FASGOON3	1BCE'	FASLP1
1BE5'	FASLP2	1BF6'	FASLP3	0837'	FAST
1C0C'	FCLP	0660'	FCOMPARE	1C04'	FCORR
1C21'	FCQUIT	0011	FCURR	1CB0'	FDGOON1
1CCB'	FDGOON2	1CFE'	FDGOON3	000F	FDICT
1C7B'	FDIV	0010	FDIVOR	1CA2'	FDLOOP1
1CB3'	FDLOOP2	1CB5'	FDLOOP3	1CBD'	FDLOOP4
1CE8'	FDNEXT	0040	FEOFFS	0000	FEXP1
0001	FEXP2	0000	FFLAG	1AF8'	FICLEAR
1A3D'	FILEBHEAD	1A10'	FILEFHEAD	19F3'	FILENAME
063D'	FIND	1620'	FINDWORD	1AF4'	FINIT
1278'	FJUMP	3C3E	FLAGS	000B	FLEN
064B'	FLOOP	1C3D'	FLT0	1C40'	FLT0LP
1C3B'	FLTERR	1BA4'	FMINUS	1C5D'	FMLOOP
1B67'	FMLOOP1	1B77'	FMLOOP2	1B91'	FMNEXT
1B83'	FMNOADD	1C4B'	FMUL	1B55'	FMULADD
1B43'	FNEG	1D0F'	FNEGATE	067D'	FNEXT1
067F'	FNEXT2	0001	FNLEN	1B48'	FNLOOP
1D18'	FNQUIT	1638'	FORGET	0133'	FORTH
0AFC'	FP0	2301	FPADMEM	0B05'	FPEXP
0ABE'	FPGOON1	0ACA'	FPGOON2	0ACE'	FPGOON3
0AD7'	FPH0	1BB1'	FPLUS	0ADC'	FPMLOOP
0AAF'	FPNT	0B10'	FPQUIT	0676'	FPRINT
15E7'	FPTR2NAME	3C00	FPWS	0007	FQUO
3C2B	FRAMES	3C51	FREEMEM	0002	FSGN
0080	FSIGN	0019	FSIZE	000D	FSTART
0657'	FTEST	0742'	FZEQLP	0740'	FZEROEQ
104B'	GETBYTE	048D'	GETFLAGS	1064'	GETFLOAT
05DF'	GETSTRING	044D'	GETVAR	1011'	GETWORD
0004	GFX	0C99'	GREATER	0CA0'	GRTRQUIT

0C56'	GT	08D2'	GTR	1019'	GWGOON
1015'	GWLOOP	101E'	GWQUIT	0460'	HERE
3C1A	HLD	0A5C'	HOLD	0A69'	HOLDQUIT
12E9'	I	0D9E'	I0NEND	11C0'	IF
128F'	IF0JUMP	0D94'	IFN0NEG	043D'	ILLOOP
0040	IMM	1160'	IMMEDIATE	0BEB'	IN
0BDB'	INKEY	3C1E	INSCRN	042F'	INSLINE
1D22'	INT	1D2B'	INTLOOP	1D45'	INTQUIT
0008	INV	0828'	INVIS	00FE	IO
12F7'	ITICK	1302'	J	15F9'	JDELOOP
132D'	JNEXT4	15FB'	JUMPDE	0009	KDN
3C27	KEYCNT	3C26	KEYCOD	0336'	KEYGET
034F'	KEYGLP	0347'	KEYGNC	0362'	KEYGNK
036B'	KEYGQU	036D'	KEYGQU2	0359'	KEYGSC
0376'	KEYTBL	0001	KLT	0F4E'	KOMMA
0003	KRT	0007	KUP	0E4B'	LAND
1361'	LBRACKET	1368'	LBREND	000A	LDL
19DD'	LDNLOOP	1316'	LEAVE	3C24	LHALF
0F36'	LHGOON	0506'	LINE	0530'	LINEERR
0508'	LINELOOP	0518'	LINENUM	0526'	LINESTR
0F2E'	LINKHERE	1670'	LIST	1708'	LISTPGM
3C13	LISTWS	1006'	LITERAL	1055'	LITFLOAT
198A'	LOAD	0002	LOK	12BD'	LOOP
133F'	LOOPADD	1350'	LOOPCMP	1358'	LOOPEND
12C1'	LOOPGOON	0E36'	LOR	1798'	LPBYTE
178B'	LPFLOAT	3C15	LPIACT	3C14	LPIBUF
3C13	LPICNT	176B'	LPIDEC	175D'	LPIINC
1764'	LPILEFT	1770'	LPINDENT	17AC'	LPLBRACKET
3C16	LPLCNT	1712'	LPLOOP	1756'	LPNEXT
1783'	LPNUMBER	17DA'	LPNXTWRD	1753'	LPOUT
17B0'	LPPTSTR	17A4'	LPSEMIS	17B2'	LPSTRING
177C'	LPWORD	13D5'	LSQRBR	0C65'	LT
098D'	LTNUM	0E60'	LXOR	0E75'	MAX
0F9C'	MCERROR	3C00	MEMBEG	0F8C'	MEMCHECK
0F8F'	MEMCHECK2	0E87'	MIN	0E8F'	MINMAX
0E95'	MINMAXEND	0DE1'	MINUS	0D61'	MOD
0D6D'	MUL	0D7A'	MULDIV	0D31'	MULDIVMOD
0A13'	NADEC	1020'	NASCII	1212'	NBEGIN
0EAB'	NCOLON	10EA'	NCOMPILER	0FD7'	NCONSTANT
0EC7'	NCREATE	106A'	NDEFINER	12A6'	NDO
10AC'	NDOESGT	0DA9'	NEGATE	11E5'	NELSE
04B9'	NEXT	04BF'	NEXTDE	04BA'	NEXTSUB
15B5'	NFA	06FD'	NFEGOON	06EF'	NFEXP
06CE'	NFGOON	06BC'	NFLOAT	06D3'	NFLOOP1
06DF'	NFLOOP2	0711'	NFQUIT	0A07'	NIBASC
11BB'	NIF	135D'	NLBRACKET	12B6'	NLOOP
12C8'	NPLUSLOOP	1383'	NPTSTR	1243'	NREPEAT
111D'	NRUNSGT	049D'	NSEMICOLON	1200'	NTHEN
09F7'	NUM	06A9'	NUMBER	071C'	NUMBERERR
0714'	NUMBERQUIT	099C'	NUMGT	09E1'	NUMS
09E3'	NUMSLP	125B'	NUNTIL	0FC4'	NVARIABLE
11CD'	NWHILE	127C'	OFFSJUMP	17CF'	OILOOP
17D4'	OIQUIT	0536'	OK	054D'	OKQUIT
0E1F'	ONEMINUS	0E09'	ONEPLUS	0BFD'	OUT
17C1'	OUTINDENT	17FB'	OUTTXT	17E4'	OUTWORD
17E1'	OUTWORDI	0912'	OVER	17F0'	OWDOXX

```
15DB'   P2AGOON         15D4'   P2ARUN          15F2'   P2NGOON
15F4'   P2NLOOP         0499'   PAD             2701    PADMEM
0925'   PICK            0B6F'   PLGOON          0B4A'   PLOT
0DD2'   PLUS            12D0'   PLUSLOOP        0B7F'   PLX0Y0
0B8C'   PLXOR           0060    PND             09B3'   PNT
180E'   PNTHL           09C3'   PNTLEFT         15C0'   PTR2ADDR
15E8'   PTR2NAME        1388'   PTSTR           084E'   PULLBC
0CF3'   PUSHDEHL        08EE'   QDUP            04F5'   QLLOOP
059B'   QLOOP           0594'   QSTART          058C'   QUERY
0099'   QUIT            04F2'   QUITLOOP        00AD'   RABORT
18EC'   RAGOON          18DF'   RALOOP          18B6'   RALOOP1
18B8'   RALOOP2         18C7'   RALOOP3         3C18    RAMTOP
3C24    RAMVAR          18F0'   RASTART         18B5'   RASYNC
18E7'   RAVERIFY        18FE'   RB8LOOP         1917'   RBDELAY
191B'   RBLOOP          0085'   RCHR7           007C'   RCHRLP
000A    RDDLEN          0004    RDDNAME         1452'   RDGOON1
147F'   RDGOON2         0004    RDNCODE         000A    RDNEND
000C    RDNNAME         0006    RDNRUN          0002    RDOCODE
0008    RDOEND          0000    RDONAME         1A74'   READHEADER
14CF'   REDEFABORT      13FD'   REDEFINE        03EE'   REMIT
03F5'   RENORM          124C'   REPEAT          1610'   RESCURR
18FB'   RETURN          0578'   RETYPE          0644'   RFIND
1307'   RGET            0055'   RGFXLP          005F'   RGFXM
003B'   RGOON           08DF'   RGT             1A95'   RHBINARY
1A9F'   RHCHECK         1AA7'   RHCLOOP         1AAE'   RHCNEXT
1AA9'   RHCSTART        1A76'   RHLOOP          0A5F'   RHOLD
0028'   RMEMLP          0933'   ROLL            1FFC'   ROMCHR
1808'   ROMTXT          010D'   ROMVAR          013A'   ROMVEND
08FF'   ROT             0859'   RPULL           085F'   RPUSH
009B'   RQUIT           04B8'   RSEMIS          04C8'   RSLNEXT
04D9'   RSLNGOON        13E1'   RSQRBR          1142'   RUNSCORR
1125'   RUNSGT          18A7'   RXALL           1911'   RXBIT
18FC'   RXBYTE          1AB6'   RXERROR         1915'   RXLEVEL
068A'   RZERO           000C    SAFETY          1934'   SAVE
139F'   SAVETEXT        0290'   SCNOCAPS        2400    SCREEN
2700    SCREND          2800    SCRMEND         0421'   SCROLLUP
3C1C    SCRPOS          1A0E'   SEMICODE        04A1'   SEMICOLON
04B6'   SEMIS           11B5'   SETCONTEXT      0A4A'   SIGN
159E'   SKIPOFFS        15A2'   SKOFFS2         15B1'   SKOGOON
04C6'   SLNEXT          0846'   SLOW            0A73'   SPACE
0A78'   SPACEQUIT       0A83'   SPACES          3C3B    SPARE
0A86'   SPCLOOP         3C28    STATIN          13B8'   STFND
3C37    STKBOT          13A1'   STLOOP          0885'   SWAP
1864'   TABIT0          1887'   TACHECK         1839'   TADEL1
1847'   TADEL2          184F'   TADEL3          185C'   TADEL4
1862'   TADEL5          188F'   TAEND           1832'   TAGOON1
1843'   TAGOON2         1837'   TALOOP1         1859'   TALOOP2
186D'   TANEXT          188A'   TASTART         1207'   THEN
0807'   TOUPPER         1897'   TRQDEL6         0E29'   TWOMINUS
0E13'   TWOPLUS         1820'   TXALL           1892'   TXRXQUIT
096E'   TYPE            0979'   TYPEDE          0C77'   UCMP
0D8C'   UDIVMOD         1D59'   UFLOAT          1D62'   UFLOOP
0C72'   ULT             0CA8'   UMUL            0CB3'   UMULLOOP
0CBE'   UMULNEXT        1263'   UNTIL           09D0'   UPNT
04E4'   USERBREAK       0FCF'   VARIABLE        3C3F    VBASE
3C33    VCONTEXT        3C31    VCURRENT        0142'   VDELAY
```

```
1967'  VERIFY          0818'     VIS             0325'     VKAGAIN
0310'  VKEY            0320'     VKNEW           0331'     VKPRESS
0332'  VKQUIT          062D'     VLIST           117D'     VOCABULARY
3C35   VOCLNK          0147'     VSCNT           0170'     VSCTRL
0176'  VSEND           0167'     VSNOGRF         016D'     VSNOINV
013A'  VSYNC           05B3'     WCLLOOP         05C6'     WGOON1
05D1'  WGOON2          11D5'     WHILE           05AB'     WORD
3C2F   XCOORD          0E2D'     XMINUS          0E17'     XPLUS
0E2E'  XPLUSMINUS      3C30      YCOORD          0688'     ZERO
0C1A'  ZEROEQ          0C3A'     ZEROGT          0C2E'     ZEROLT


No Fatal error(s)


#
3C35   VOCLNK          0147'     VSCNT           0170'     VSCTRL
0176'  VSEND           0167'     VSNOG
```

Z80 OPcodes

(a private Chart for)

Groups &Timmings

Relative CPU speeds, by nominal MHz

(values to consider when attempting 8bit CPUs emulation)

!! Motorola nominal MHz half of internal Clock, or Intel equivalents. We will consider INTERNAL Clocks !!
!! As previous CPUs are 8bit, with the 8086, CPUs are 16bit (or 32 bit), doubling actual speed !!
!! Rule of 50x (20x+30x) for 16bit emulation of 8 bit... is tools and structures dependent !!
!! Motorola 32bit series, 68000 are not shown. They evolved quickly, later 'stolen' !!

(/6809)	Efficiency	MHz => Factor	
8080	0.18	3 MHz => 0.57	...
6800	0.21	(4)MHz => 0.84	<u>Nominal 2MHz</u>
Z-80	0.28	4 MHz => 1.12	Super 8080 (with extensions)
6809	**0.50**	(4)MHz => 2.00	<u>Nominal 2MHz</u> (+16bit Ops)
8086	0.80	5 MHz => 4.00	Fully 16 bits = 2x0.40
286	1.28	8 MHz => 10.24	Fully 16 bits = 2x0.64 (+32bit Ops)

(/8086)	Efficiency	MHz => Factor	
Z80	0.35	4 MHz => 1.4	8 bit only
6809	0.63	(4)MHz => 2.5	<u>Nominal 2MHz</u> (+ 16bit OPs)
8086	**1.00**	**5 MHz => 5.0**	Full 16bit (with a few 8bit OPs)
286	1.60	10 MHz => 16.	...
386	2.56	16 MHz => 41.	(29x Z80/4MHz) – ASM only
486	5.12	33 MHz => 169.	(120x Z80/4MHz) – Any compiler
586	7.68	90 MHz => 691.	(495x Z80/4MHz) – Even bad emulation

(/386)	Efficiency	MHz => Factor	
Z80	0.14	4 MHz => 0.55	
6809	0.24	(4)MHz => 0.97	<u>Nominal 2MHz</u> (+ 16bit OPs)
8086	0.39	9 MHz => 3.51	(6xZ80/4MHz)
i286	0.63	16 MHz => 10.08	(18x Z80/4MHz) – Tricky...
i386	**1.00**	**33 MHz => 33.75**	(62x Z80/4MHz) – A good compiler
i486	2.00	50 MHz => 100.0	(183x Z80/4MHz) – Even bad emulation
i586	3.00	100 MHz => 300.0	

Z80 Opcodes: Groups and Timmings

Compilation and Format: Dutra de Lacerda,2017

(Initial data based on J.G.Harston and Others)

MAIN

A search for patterns and timmings (for Hand-Assembly and for Groups Emulation)

Manage: * nn (--) RR n *

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
00	●NOP 4	LD BC,nn 10	LD (BC),A 7	INC BC 6	INC B 4	DEC B 4	LD B,n 7	RLCA 4	0x
08	●EX AF,AF 4	ADD HL,BC 11	LD A,(BC) 7	DEC BC 6	INC C 4	DEC C 4	LD C,n 7	RRCA 4	
10	DJNZ d 13	LD DE,nn 10	LD (DE),A 7	INC DE 6	INC D 4	DEC D 4	LD D,n 7	RLA 4	1x
	8								
18	JR d 12	ADD HL,DE 11	LD A,(DE) 7	DEC DE 6	INC E 4	DEC E 4	LD E,n 7	RRA 4	
	7								
20	JR NZ,d 12	LD HL,nn 10	LD (nn),HL 16	INC HL 6	INC H 4	DEC H 4	LD H,n 7	DAA 4	2x
	7								
28	JR Z,d 12	ADD HL,HL 11	LD HL,(nn) 16	DEC HL 6	INC L 4	DEC L 4	LD L,n 7	CPL 4	
	7								
30	JR NC,d 12	LD SP,nn 10	LD (nn),A 13	INC SP 6	INC (HL) 11	DEC (HL) 11	LD (HL),n 10	SCF 4	3x
	7								
38	JR C,d 12	ADD HL,SP 11	LD A,(nn) 13	DEC SP 6	INC A 4	DEC A 4	LD A,n 7	CCF 4	
	7								
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

Loads:

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
40	LD B,B 4	LD B,C 4	**LD B,D** 4	LD B,E 4	**LD B,H** 4	LD B,L 4	LD B,(HL) 7	LD B,A 4	4x
48	LD C,B 4	LD C,C 4	LD C,D 4	**LD C,E** 4	LD C,H 4	**LD C,L** 4	LD C,(HL) 7	LD C,A 4	
50	**LD D,B** 4	LD D,C 4	LD D,D 4	LD D,E 4	**LD D,H** 4	LD D,L 4	LD D,(HL) 7	LD D,A 4	5x
58	LD E,B 4	**LD E,C** 4	LD E,D 4	LD E,E 4	LD E,H 4	**LD E,L** 4	LD E,(HL) 7	LD E,A 4	
60	**LD H,B** 4	LD H,C 4	**LD H,D** 4	LD H,E 4	LD H,H 4	LD H,L 4	LD H,(HL) 7	LD H,A 4	6x
68	LD L,B 4	**LD L,C** 4	LD L,D 4	**LD L,E** 4	LD L,H 4	LD L,L 4	LD L,(HL) 7	LD L,A 4	
70	LD (HL),B 7	LD (HL),C 7	LD (HL),D 7	LD (HL),E 7	LD (HL),H 7	LD (HL),L 7	●**HALT** 4	LD (HL),A 7	7x
78	LD A,B 4	LD A,C 4	LD A,D 4	LD A,E 4	LD A,H 4	LD A,L 4	LD A,(HL) 7	LD A,A 4	
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

Arithm:

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
80	ADD A,B 4	ADD A,C 4	ADD A,D 4	ADD A,E 4	ADD A,H 4	ADD A,L 4	ADD A,(HL) 7	ADD A,A 4	8x
88	ADC A,B 4	ADC A,C 4	ADC A,D 4	ADC A,E 4	ADC A,H 4	ADC A,L 4	ADC A,(HL) 7	ADC A,A 4	
90	SUB A,B 4	SUB A,C 4	SUB A,D 4	SUB A,E 4	SUB A,H 4	SUB A,L 4	SUB A,(HL) 7	SUB A,A 4	9x
98	SBC A,B 4	SBC A,C 4	SBC A,D 4	SBC A,E 4	SBC A,H 4	SBC A,L 4	SBC A,(HL) 7	SBC A,A 4	
A0	AND B 4	AND C 4	AND D 4	AND E 4	AND H 4	AND L 4	AND (HL) 7	AND A 4	Ax
A8	XOR B 4	XOR C 4	XOR D 4	XOR E 4	XOR H 4	XOR L 4	XOR (HL) 7	XOR A 4	
B0	OR B 4	OR C 4	OR D 4	OR E 4	OR H 4	OR L 4	OR (HL) 7	OR A 4	Bx
B8	CP B 4	CP C 4	CP D 4	CP E 4	CP H 4	CP L 4	CP (HL) 7	CP A 4	
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

Control: ?ret Pop ?Jp (--) ?call Push A,n rst

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
	5		10		10				
C0	RET NZ 11	POP BC 10	JP NZ,nn 10	JP nn 10	CALL NZ,nn 17	PUSH BC 11	ADD A,n 7	RST &00 11	Cx
	5		10		10				
C8	RET Z 11	● **RET** 10	JP Z,nn 10	● ***>>CB Op:*** 4	CALL Z,nn 17	● **CALL nn** 17	ADC A,n 7	RST &08 11	
	5		10		10				
D0	RET NC 11	POP DE 10	JP NC,nn 10	OUT (n),A 11	CALL NC,nn 17	PUSH DE 11	SUB A,n 7	RST &10 11	Dx
	5		10		10				
D8	RET C 11	● **EXX** 4	JP C,nn 10	IN A,(n) 11	CALL C,nn 17	● ***DD => iX***	SBC A,n 7	RST &18 11	
	5		10		10				
E0	RET PO 11	POP HL 10	JP PO,nn 10	EX (SP),HL 19	CALL PO,nn 17	PUSH HL 11	AND n 7	RST &20 11	Ex
	5		10		10				
E8	RET PE 11	● **JP (HL)** 10	JP PE,nn 10	●**EX DE,HL** 4	CALL PE,nn 17	● ***>>ED Op:*** 4	XOR n 7	RST &28 11	
	5		10		10				
F0	RET P 11	POP AF 10	JP P,nn 10	DI 4	CALL P,nn 17	PUSH AF 11	OR n 7	RST &30 11	Fx
	5		10		10				
F8	RET M 11	● **LD SP,HL** 6	JP M,nn 10	EI 4	CALL M,nn 17	● ***FD => iY***	CP n 7	RST &38 11	
	5		10		10				
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

Usual Clocks: Dispatch = 2T Reg Oper = +2 (ind) = +3
ReadExtraByte = +3 WriteExtraByte = +3/+4

Z80 Opcodes: Groups and Timmings

Compilation and Format: Dutra de Lacerda,2017
(Initial data based on J.G.Harston and Others)

BITS = **CB** prefix

All Ops #T = 4+$ = (when total not mentioned, assume 8 or the first in column)
Inexistant Ops act as usual Main Table Ops, DF/FD ignored, 4T already run (added)

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
00	RLC B 8	RLC C 8	RLC D 8	RLC E 8	RLC H 8	RLC L 8	RLC (HL) 15	RLC A 8	0x
08	RRC B	RRC C	RRC D	RRC E	RRC H	RRC L	RRC (HL) 15	RRC A	
10	RL B	RL C	RL D	RL E	RL H	RL L	RL (HL) 15	RL A	1x
18	RR B	RR C	RR D	RR E	RR H	RR L	RR (HL) 15	RR A	
20	SLA B	SLA C	SLA D	SLA E	SLA H	SLA L	SLA (HL) 15	SLA A	2x
28	SRA B	SRA C	SRA D	SRA E	SRA H	SRA L	SRA (HL) 15	SRA A	
30	sls B	sls c	sls D	sls E	sls H	sls L	sls (hl) 15	sls a	3x
38	SRL B	SRL C	SRL D	SRL E	SRL H	SRL L	SRL (HL) 15	SRL A	
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
40	BIT 0,B 8	BIT 0,C 8	BIT 0,D 8	BIT 0,E 8	BIT 0,H 8	BIT 0,L 8	BIT 0,(HL) 15	BIT 0,A 8	4x
48	BIT 1,B	BIT 1,C	BIT 1,D	BIT 1,E	BIT 1,H	BIT 1,L	BIT 1,(HL) 15	BIT 1,A	
50	BIT 2,B	BIT 2,C	BIT 2,D	BIT 2,E	BIT 2,H	BIT 2,L	BIT 2,(HL) 15	BIT 2,A	5x
58	BIT 3,B	BIT 3,C	BIT 3,D	BIT 3,E	BIT 3,H	BIT 3,L	BIT 3,(HL) 15	BIT 3,A	
60	BIT 4,B	BIT 4,C	BIT 4,D	BIT 4,E	BIT 4,H	BIT 4,L	BIT 4,(HL) 15	BIT 4,A	6x
68	BIT 5,B	BIT 5,C	BIT 5,D	BIT 5,E	BIT 5,H	BIT 5,L	BIT 5,(HL) 15	BIT 5,A	
70	BIT 6,B	BIT 6,C	BIT 6,D	BIT 6,E	BIT 6,H	BIT 6,L	BIT 6,(HL) 15	BIT 6,A	7x
78	BIT 7,B	BIT 7,C	BIT 7,D	BIT 7,E	BIT 7,H	BIT 7,L	BIT 7,(HL) 15	BIT 7,A	
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
80	RES 0,B 8	RES 0,C 8	RES 0,D 8	RES 0,E 8	RES 0,H 8	RES 0,L 8	RES 0,(HL) 15	RES 0,A 8	8x
88	RES 1,B	RES 1,C	RES 1,D	RES 1,E	RES 1,H	RES 1,L	RES 1,(HL) 15	RES 1,A	
90	RES 2,B	RES 2,C	RES 2,D	RES 2,E	RES 2,H	RES 2,L	RES 2,(HL) 15	RES 2,A	9x
98	RES 3,B	RES 3,C	RES 3,D	RES 3,E	RES 3,H	RES 3,L	RES 3,(HL) 15	RES 3,A	
A0	RES 4,B	RES 4,C	RES 4,D	RES 4,E	RES 4,H	RES 4,L	RES 4,(HL) 15	RES 4,A	Ax
A8	RES 5,B	RES 5,C	RES 5,D	RES 5,E	RES 5,H	RES 5,L	RES 5,(HL) 15	RES 5,A	
B0	RES 6,B	RES 6,C	RES 6,D	RES 6,E	RES 6,H	RES 6,L	RES 6,(HL) 15	RES 6,A	Bx
B8	RES 7,B	RES 7,C	RES 7,D	RES 7,E	RES 7,H	RES 7,L	RES 7,(HL) 15	RES 7,A	
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
C0	SET 0,B 8	SET 0,C 8	SET 0,D 8	SET 0,E 8	SET 0,H 8	SET 0,L 8	SET 0,(HL) 15	SET 0,A 8	Cx
C8	SET 1,B	SET 1,C	SET 1,D	SET 1,E	SET 1,H	SET 1,L	SET 1,(HL) 15	SET 1,A	
D0	SET 2,B	SET 2,C	SET 2,D	SET 2,E	SET 2,H	SET 2,L	SET 2,(HL) 15	SET 2,A	Dx
D8	SET 3,B	SET 3,C	SET 3,D	SET 3,E	SET 3,H	SET 3,L	SET 3,(HL) 15	SET 3,A	
E0	SET 4,B	SET 4,C	SET 4,D	SET 4,E	SET 4,H	SET 4,L	SET 4,(HL) 15	SET 4,A	Ex
E8	SET 5,B	SET 5,C	SET 5,D	SET 5,E	SET 5,H	SET 5,L	SET 5,(HL) 15	SET 5,A	
F0	SET 6,B	SET 6,C	SET 6,D	SET 6,E	SET 6,H	SET 6,L	SET 6,(HL) 15	SET 6,A	Fx
F8	SET 7,B	SET 7,C	SET 7,D	SET 7,E	SET 7,H	SET 7,L	SET 7,(HL) 15	SET 7,A	
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

Usual Clocks: Dispatch = 2T Reg Oper = +2 (ind) = +3
ReadExtraByte = +3 WriteExtraByte = +3/+4

Z80 Opcodes: Groups and Timmings

Compilation and Format: Dutra de Lacerda,2017

(Initial data based on J.G.Harston and Others)

Indexed = **DD/FD** prefix (for iX/iY)

All Ops #T: +4 (HL replacement mode)

Inexistant Ops act as usual Main Table Ops, DF/FD ignored, 4T already run (added)

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
00									0x
08		ADD XY,BC 15							
10									1x
18		ADD XY,DE 15							
20		LD XY,nn 14	LD (nn),XY 20	INC XY 10	INC XYh 8	DEC XYh 8	LD XYh,n 11		2x
28		ADD XY,XY 15	LD XY,(nn) 20	DEC XY 10	INC XYl 8	DEC XYl 8	LD XYl,n 11		
30					INC (XY+d) 26	DEC (XY+d) 26	LD(XY+d),n 19		3x
38		ADD XY,SP 15							
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
40					LD B,XYh 8	LD B,XYl 8	LD B,(XY+d)19		4x
48					LD C,XYh 8	LD C,XYl 8	LD C,(XY+d)19		
50					LD D,XYh 8	LD D,XYl 8	LD D,(XY+d)19		5x
58					LD E,XYh 8	LD E,XYl 8	LD E,(XY+d)19		
60	LD XYh,B 8	LD XYh,C 8	LD XYh,D 8	LD XYh,E 8	LD XYh,XYh 8	LD XYh,XYl 8	LD H,(XY+d)19	LD XYh,A 8	6x
68	LD XYl,B 8	LD XYl,C 8	LD XYl,D 8	LD XYl,E 8	LD XYl,XYh 8	LD XYl,XYl 8	LD L,(XY+d)19	LD XYl,A 8	
70	LD(XY+d),B 19	LD(XY+d),C 19	LD(XY+d),D 19	LD(XY+d),E 19	LD(XY+d),H 19	LD(XY+d),L 19		LD(XY+d),A 19	7x
78					LD A,XYh 8	LD A,XYl 8	LD A,(XY+d)19		
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
80					ADD A,XYh 8	ADD A,XYl 8	ADD A(XY+d)19		8x
88					ADC A,XYh 8	ADC A,XYl 8	ADC A(XY+d)19		
90					SUB A,XYh 8	SUB A,XYl 8	SUB A(XY+d)19		9x
98					SBC A,XYh 8	SBC A,XYl 8	SBC A(XY+d)19		
A0					AND XYh 8	AND XYl 8	AND (XY+d) 19		Ax
A8					XOR XYh 8	XOR XYl 8	XOR (XY+d) 19		
B0					OR XYh 8	OR XYl 8	OR (XY+d) 19		Bx
B8					CP XYh 8	CP XYl 8	CP (XY+d) 19		
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
C0									Cx
C8									
D0									Dx
D8									
E0		POP XY 14		EX (SP),XY 23		PUSH XY 15			Ex
E8		JP (XY) 8							
F0									Fx
F8		LD SP,XY 10							
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

Usual Clocks: Dispatch = 2T Reg Oper = +2 (ind) = +3
ReadExtraByte = +3 WriteExtraByte = +3/+4

Z80 Opcodes: Groups and Timmings

Compilation and Format: Dutra de Lacerda,2017

(Initial data based on J.G.Harston and Others)

Indexed,Bit = **DD/FD, CB** prefix (for iX/iY)

All Ops #T: 4+Work = (the first in column, or assume group column 6) Format is DD|FD CB n Opcode
N follows CB with XY modifier:'n' kept as prefetched 3th byte (2 bytes read ahead)
Reason is N is an inconstant argument, not dealt by ad-hoc operations

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
00	rlc (xy+n),b	rlc (xy+n),c	rlc (xy+n),d	rlc (xy+n),e	RLC XYh 16	RLC XYl 16	RLC (XY+n) 23	rlc (xy+n,a)	0x
08	rrc (xy+n),b	rrc (xy+n),c	rrc (xy+n),d	rrc (xy+n),e	RRC XYh	RRC XYl	RRC (XY+n)	rrc (xy+n),a	
10	rl (xy+n),b	rl (xy+n),c	rl (xy+n),d	rl (rc+n),e	RL XYh	RL XYl	RL (XY+n)	rl (xy+n),a	1x
18	rr (xy+n),b	rr (xy+n),c	rr (xy+n),d	rr (xy+n),e	RR XYh	RR XYl	RR (XY+n)	rr (xy+n),a	
20	sla (xy+n),b	sla (xy+n),c	sla (xy+n),d	sla (xy+n),e	SLA XYh	SLA XYl	SLA (XY+n)	sla (xy+n),a	2x
28	sra (xy+n),b	sra (xy+n),c	sra (xy+n),d	sra (xy+n),e	SRA XYh	SRA XYl	SRA (XY+n)	sra (xy+n),a	
30	sls (xy+n),b	sls (xy+n),c	sls (xy+n),d	sls (xy+n),e	sls xyh	sls xyl	sls (xy+n)	sls (xy+n),a	3x
38	srl (xy+n),b	srl (xy+n),c	srl (xy+n),d	srl (xy+n),e	SRL XYh	SRL XYl	SRL (XY+n)	srl (xy+n),a	
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
40	bit 0,(xy+n)	bit 0,(xy+n)	bit 0,(xy+n)	bit 0,(xy+n)	BIT 0,XYh 16	BIT 0,XYl 16	BIT0,(XY+n)23	bit 0,(xy+n)	4x
48	bit 1,(xy+n)	bit 1,(xy+n)	bit 1,(xy+n)	bit 1,(xy+n)	BIT 1,XYh	BIT 1,XYl	BIT1,(XY+n)	bit 1,(xy+n)	
50	bit 2,(xy+n)	bit 2,(xy+n)	bit 2,(xy+n)	bit 2,(xy+n)	BIT 2,XYh	BIT 2,XYl	BIT2,(XY+n)	bit 2,(xy+n)	5x
58	bit 3,(xy+n)	bit 3,(xy+n)	bit 3,(xy+n)	bit 3,(xy+n)	BIT 3,XYh	BIT 3,XYl	BIT3,(XY+n)	bit 3,(xy+n)	
60	bit 4,(xy+n)	bit 4,(xy+n)	bit 4,(xy+n)	bit 4,(xy+n)	BIT 4,XYh	BIT 4,XYl	BIT4,(XY+n)	bit 4,(xy+n)	6x
68	bit 5,(xy+n)	bit 5,(xy+n)	bit 5,(xy+n)	bit 5,(xy+n)	BIT 5,XYh	BIT 5,XYl	BIT5,(XY+n)	bit 5,(xy+n)	
70	bit 6,(xy+n)	bit 6,(xy+n)	bit 6,(xy+n)	bit 6,(xy+n)	BIT 6,XYh	BIT 6,XYl	BIT6,(XY+n)	bit 6,(xy+n)	7x
78	bit 7,(xy+n)	bit 7,(xy+n)	bit 7,(xy+n)	bit 7,(xy+n)	BIT 7,XYh	BIT 7,XYl	BIT7,(XY+n)	bit 7,(xy+n)	
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
80	res0,(xy+n),b	res0,(xy+n),c	res0,(xy+n),d	res0,(xy+n),e	RES 0,XYh 16	RES 0,XYl 16	RES0,(XY+n)23	res0,(xy+n),a	8x
88	res1,(xy+n),b	res1,(xy+n),c	res1,(xy+n),d	res1,(xy+n),e	RES 1,XYh	RES 1,XYl	RES1,(XY+n)	res1,(xy+n),a	
90	res2,(xy+n),b	res2,(xy+n),c	res2,(xy+n),d	res2,(xy+n),e	RES 2,XYh	RES 2,XYl	RES2,(XY+n)	res2,(xy+n),a	9x
98	res3,(xy+n),b	res3,(xy+n),c	res3,(xy+n),d	res3,(xy+n),e	RES 3,XYh	RES 3,XYl	RES3,(XY+n)	res3,(xy+n),a	
A0	res4,(xy+n),b	res4,(xy+n),c	res4,(xy+n),d	res4,(xy+n),e	RES 4,XYh	RES 4,XYl	RES4,(XY+n)	res4,(xy+n),a	Ax
A8	res5,(xy+n),b	res5,(xy+n),c	res5,(xy+n),d	res5,(xy+n),e	RES 5,XYh	RES 5,XYl	RES5,(XY+n)	res5,(xy+n),a	
B0	res6,(xy+n),b	res6,(xy+n),c	res6,(xy+n),d	res6,(xy+n),e	RES 6,XYh	RES 6,XYl	RES6,(XY+n)	res6,(xy+n),a	Bx
B8	res7,(xy+n),b	res7,(xy+n),c	res7,(xy+n),d	res7,(xy+n),e	RES 7,XYh	RES 7,XYl	RES7,(XY+n)	res7,(xy+n),a	
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
C0	set0,(xy+n),b	set0,(xy+n),c	set0,(xy+n),d	set0,(xy+n),e	SET 0,XYh 16	SET 0,XYl 16	SET0,(XY+n)23	set0,(xy+n),a	Cx
C8	set1,(xy+n),b	set1,(xy+n),c	set1,(xy+n),d	set1,(xy+n),e	SET 1,XYh	SET 1,XYl	SET1,(XY+n)	set1,(xy+n),a	
D0	set2,(xy+n),b	set2,(xy+n),c	set2,(xy+n),d	set2,(xy+n),e	SET 2,XYh	SET 2,XYl	SET2,(XY+n)	set2,(xy+n),a	Dx
D8	set3,(xy+n),b	set3,(xy+n),c	set3,(xy+n),d	set3,(xy+n),e	SET 3,XYh	SET 3,XYl	SET3,(XY+n)	set3,(xy+n),a	
E0	set4,(xy+n),b	set4,(xy+n),c	set4,(xy+n),d	set4,(xy+n),e	SET 4,XYh	SET 4,XYl	SET4,(XY+n)	set4,(xy+n),a	Ex
E8	set5,(xy+n),b	set5,(xy+n),c	set5,(xy+n),d	set5,(xy+n),e	SET 5,XYh	SET 5,XYl	SET5,(XY+n)	set5,(xy+n),a	
F0	set6,(xy+n),b	set6,(xy+n),c	set6,(xy+n),d	set6,(xy+n),e	SET 6,XYh	SET 6,XYl	SET6,(XY+n)	set6,(xy+n),a	Fx
F8	set7,(xy+n),b	set7,(xy+n),c	set7,(xy+n),d	set7,(xy+n),e	SET 7,XYh	SET 7,XYl	SET7,(XY+n)	set7,(xy+n),a	
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

Usual Clocks: Dispatch = 2T Reg Oper = +2 (ind) = +3
ReadExtraByte = +3 WriteExtraByte = +3/+4

Z80 Opcodes: Groups and Timmings

Compilation and Format: Dutra de Lacerda,2017

(Initial data based on J.G.Harston and Others)

<u>Extra</u> = **ED** Prefix

#T: 4+Work = (when not mentioned assume the first in column)

Inexistant Ops generate NOP, with 4Ts (unofficial OPs are in small caps)

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
00									0x
08									
10									1x
18									
20									2x
28									
30									3x
38									

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
40	IN B,(C) 12	OUT (C),B 12	SBC HL,BC 15	LD (nn),BC 20	NEG 8	RETN 14	IM 0 8	LD I,A 9	4x
48	IN C,(C) 12	OUT (C),C 12	ADC HL,BC 15	LD BC,(nn) 20	neg	RETI 14	im 0	LD R,A 9	
50	IN D,(C) 12	OUT (C),D 12	SBC HL,DE 15	LD (nn),DE 20	neg	retn	IM 1 8	LD A,I 9	5x
58	IN E,(C) 12	OUT (C),E 12	ADC HL,DE 15	LD DE,(nn) 20	neg	retn	IM 2 8	LD A,R 9	
60	IN H,(C) 12	OUT (C),H 12	SBC HL,HL 15	LD (nn),HL 20	neg	retn	im 0	RRD 18	6x
68	IN L,(C) 12	OUT (C),L 12	ADC HL,HL 15	LD HL,(nn) 20	neg	retn	im 0	RLD 18	
70	IN F,(C) 12	OUT (C),F 12	SBC HL,SP 15	LD (nn),SP 20	neg	retn	im 1	ld i,i	7x
78	IN A,(C) 12	OUT (C),A 12	ADC HL,SP 15	LD SP,(nn) 20	neg	retn	im 2	ld r,r	

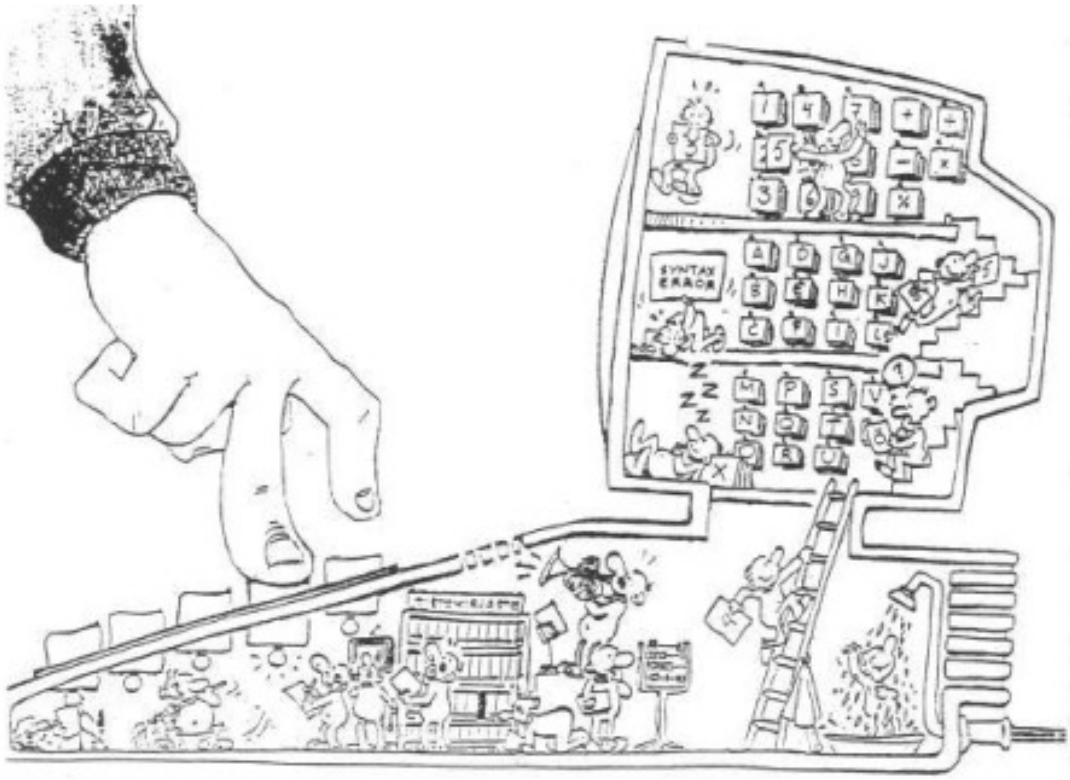
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
80									8x
88									
90									9x
98									
A0	LDI 16	CPI 16	INI 16	OTI 16					Ax
A8	LDD 16	CPD 16	IND 16	OTD 16					
B0	LDIR 21 -16	CPIR 21 -16	INIR 21 -16	OTIR 21 -16					Bx
B8	LDDR 21 -16	CPDR 21 -16	INDR 21 -16	OTDR 21 -15					

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
C0									Cx
C8									
D0									Dx
D8									
E0									Ex
E8									
F0									Fx
F8									

Usual Clocks: Dispatch = 2T Reg Oper = +2 (ind) = +3

ReadExtraByte = +3 WriteExtraByte = +3/+4

Volume #3 - Forth Ingenuity



Tiny 'Worfs', at work. *(Picture by unknown artist)*

Forth Architectures

Choices and Compromises

= Volume 3 =

Bits of Forth Internals

•

Details:

Vol#3 Author : Dutra de Lacerda
Document Date : 2022/Sept @LuxBonna, Lusitanea by the Sea.
Building Tools: An ultra-slim, lousy keyboard, small screen, dangerous touchpad.
Language used : International English, form less irrelevant, weaved as a patches blanket.

Legal Status, as enforced wherever GPL is applied or adapted:

Availability:

While now residing of https://t.me/JupiterAce the very last ACE-ROM_Doc_Prj edition can be found at https://drive.google.com/file/d/1ykjRsfCsfSOKw1YcOH6SyDjZT8hwUxtn/view?usp=sharing

Present cover:

By unknown artist, it may be replaced later.

Original cover:

The Forth Creation Myth

In the beginning...
there was the One. And uCode was with him.
On the 2th, uCode separated Code from Data,
each a place of diference.
On the 3nd phase, CFAs were made to go
beyond uCode, so all could grow.
On the 4th, 'Enter' was placed on CFAs and
the One gave him a similar, a Complement.
On the 5th, NEXT assisted uCPUs to keep the code.
Whispering tinyness forth, to Moore worlds.
On the 6th, DOES> come to help spread CFAs.
With this variety phase, Enter+Exit ruled.
On this 7th, Code was good. And the Architect
rested nearby. For Light Tyny Environments.
... New Worlds with COLOR-made uCode. Again.

Original text (by Dutra de Lacerda)
designed for "An ACE back ,and Forth" - Book#3

A few End-Of-Page taglines, to temper page reading:
To teach, is to point. Pointers available, not solutions.
To own is to do (understand)... Never tag or attribution.
Tags are just tags... Good and bad, all do have a chance.

Contents

BOOK III - Bits of Forth Internals

A quick intro

As previously stated:

I can say I wished to have by then, enough information available.
Anyway, personal experience can never be transmitted... Described, maybe.
Transmitted, no... It's personal. What we can do, is to share pointers. *We'll try.*

In a Short Presentation

This Volume is NOT to ‘teach’ how to build a Forth compiler engine. There are enough already.
(Some are not really Forth dua an absence of DOES> . So, we'll pay DOES> more attention.)
We focus on the overlooked... We point, we do not tell. So each can make his own walk.

In the following pages, you'll find perspectives. These maybe exposed differently than the usual.
Most represent a synthesis of what was observed and dispersed, an attempt to fetch the Essence.
Not ever trying to transform roosters into eagles (that does not work, nor would it be wise).

Beside, the competition in pretending such absurd has given us too much inflation...

> *‘In-ducation’ (sic) suggest that, into the great delusion of our Era:*
> *That acceptance is knowledge. Even worse, that knowledge is understanding.*
> *Considering that, we can only very rarely say “I know”. That's not even relevant.*

What matters

(we are confident on that) is **an attitude of enquire**.
... But also, of discontentment. A will to go forward.

On the endless story of ourselves we travel and never reach. We may stop, say to be 'there'.
We ‘are’ not. Are overlooking Life is the travel. Not the places passing by, and then left.

Long time ago we've tried, whenever possible, to transmit experience. More than just How.
In other words, the attitude behind. The taste of to discover what's under the surface.
There are no definitive answers for anything, just descriptions. With luck, pointers.

Plainly stated, we here attempt to pinpoint WHY of Forth is constructed as is.
For how to use Forth try the superb four books made available by “Forth Inc.”

Thus:

Hope this texts may do the same, maybe enjoyable (!?)
Hope you'll never be content, may always look a bit further.
Hope you'll can be on a constant ‘become’, never a fixed 'tagged'.

There may be some Easter-Eggs to find. *An old tradition (even found inside official law books).*

Chapter 1 - Inner Concepts

This is a reminder for what naturally follows.

Back in Ancient Greece, Algorithmica extended Greek cuisine to Mathematica.
It delivered new recipes for ancient problems mentioned in Alexandria Library. As the Great Common Divisor.
Or the Egyptian Triangle Theorem we call 'from Pitagoras', just because. (We know the Sumerians had demonstrated it, but saying otherwise pleases our ignorance). Or calculating Pi interactively.
... Just to mention a few.

```
        Simple, common Recipes
        ----------------------------------------------------------------
            |   Step   |   Step   |  Step   |   Step    |   Step   |   ->
        ----------------------------------------------------------------
```

What is program? Back in Ancient Greece, maybe from there, more likely trough Alexandria Library (of more Ancient texts), 'Algorithmica' used formal tests, going back or forward on a sequence.
Tests had replaced cuisine attention. It worked well.

```
        Simple Algorithmic recipes
                 /----------------------\
                v       backwards        ^
        ----------------------------------------------------------------
            |   Step   |   Step   |   TEST  |   Step    |   Step   |   ->
        ----------------------------------------------------------------
                                             v    forward    ^
                                              \-------------/
```

Forward to recent times, Babbage has design it. Much later, Turing and K.Zuze made it happen.
Algorithmica was recovered. Made mechanical, as the 'Bomb' or the Z1/Z2. The 'Universal Machine".
become electronic, failing to bugs (mostly ants). This until the Roswell incident and its practical recipe to working transistors. *(Since 1903, Galena crystals casually gave rare, unstable transistors.)*

In a few years, printed circuits (a hand of a few transistors) have grown exponentially from pairs to dozens, to hundreds then thousands. Later millions and billions. Yet the methods for those new Universal Machines was kept similar (e*xperts know more and more of less and less*) and crude.

```
        Simple CPU programm
                           /--------- PC -m ---------\
                          v                           ^
        ----------------------------------------------------------------------------
            |   Intr   |   Instr  |   COMP  :   DISP    |   Instr   |   Instr  |  ->
        ----------------------------------------------------------------------------
                                                       v              ^
                                                        \--- PC +n ---/
```

Instructions varied, every producer had his own ideas, changed them moved then to internal tables.
These tables made uCode, so new Operations could replace the Mainframe builder standard.
This allowed to build instructions, a capacity lost on uCPUs.

Middle term, Forth was implemented as a uCode Structured Assembler. (Not shown here, too many OPs)
Not limited to Compare, then Jump (here or there). But Structured Control OPs all CPUs should have.
Memories stayed, delayed by cheap uCPUs... Only decades later a few CPUs introduced structured OPs.

With Forth emulating uCode (the primaries), uCPUs got those benefits with little cost.
Marketing and quick profit kept problematic flow charts (problems are usually profitable):
- A crude batch commands engine, so dumb it is no longer. Only the name survives, on travesties.

No compiler was ever as small and natural, as the elegant solutions of Forth (Virtual or Native).

Step by Step

A) Sequencing, is just going NEXT: As CPU, one instruction-word after another, Step-by Step.
Original FORTH started as Sequencer (NEXT) and a Threader (Enter and Exit) these last replacing 'call/ret'.
NEXT is 'the' Dispatcher with a simple cycle: Fetch, Execute, repeat. But a 'CALL' to thread is a CPU-Op!

Note: T*here are 3 types of code: Compiled Native, Compiled Threaded, and Interpreted (just a batch).*
Note: Most professional FORTHs these days, are Compiled. (Forth is rational enough to simplify it).
While most language compilers are (now) usually threaded, as their development need to be simpler.

The CPU 'hard' Dispatcher actions:

```
  1. Fetch   : OP = [PC++]   ... Manage the sequence
  2. Execute : OP behaviour  ... Run the present instruction (dispatch)
  3. Proceed : Do it again   ... Repeat this short, closed process
```

CPUs were not limited to what people see there, or manuals say. uCPUs brought simplification/limitation.
These no longer allow OPs definition (called uCode), reason why FORTH was faster than Assembler on CPUs
as the PDP series: Forth become a new Assembler on them. A quasi-exception is the 6502 (Page Zero).

We must insist on this: That while software-dispatching cannot (on uCPUs) be native, its
speed varies with CPUs. On uCoded-CPUs there was no difference. That's how Forth was born.
This to say Software-Dispatching is RELATIVE to the uCPU (8080 and Z80 being very unfriendly).
HOW to Dispatch on uCPUs later gave other solutions, in adaptation, and keep Forth many benefits.

B) The FORTH 'soft' Sequencer+Dispatcher runs CFAs. These determine what will be done.

In the beginning there was uCode, and the ucode was good. It was efficient and coherent.
On the 2nd phase, CFAs were made to go beyond uCode. On the 3th phase CFA's separated Code and Data.
On the 4th, 'Enter' was placed on CFAs and requested its pair to make him company, to build Moore words.
On the 5th, Enter and Exit governed all codes and Var Words. On the 6th, DOES> come to help spread CFAs.
With this variety phase, Enter+Exit ruled. Then, Code was a delight... The Architect finally rested.

Sequencer is based on "Instr Pointer", Dispatcher runs CFAs. CFAs are self-suficient.
Sequence Instruction Pointer is named **IP**. It points an instruction CFA (type) to be executed.

An auxiliary register may be needed to execute the present **W**ork **S**tep **T**ype... It's called **W**.

IP and **W** are Forth internal registers, not CPU registers. These dispatch Forth sequences, and data.

<u>Note:</u> W*hen not a Primary word, W points the word Type. I.E. code to manage its data type.*
What follows (W++) is that Word own data (ParamList): A variable or a Secondary.

Thus, FORTH Sequencer(dispatcher) routine is simply called **NEXT**, including (followed by) the **RUN** routine.

```
  1. Next    : W = [IP++]     ... followed by Run
  2. Run     : Jump [W++]     ... To Code, managing a word type. (These responsible for its own return)
  N. Proceed: Do it again     ... Code executed repeats the process (after return)
```

- We mentioned that W then contains an indirect address to a Word type: code, words, data).
  This single 'type' is what makes Forth to be Forth. What unifies words in/as a concept.
  Later *W++ results of reading W, then pointing whatever follows (data the type deals with)*
- As on any routine, for the 'Proceed' step is responsibility of the word (or its type, if data).
  Every FORTH instruction ends with running NEXT, to continue the Dispatch Process.
  Ticking indefinitely, Next after Next... What about calls?

None above is a threader (a call)... What follows is:

User-built words will use another instruction to Enter(dive-in) a pack of instructions.
Ending with yet another to Exit(dive-out) back to the caller. Threading is assured.

On ASM, when calling a routine, we are threading voluntarily. On Forth it's automatic.
This is better understood with an example (before imaging it)...

: 2* DUP + **;** A word is created, started by ': name', then threading words together.
That creates a 'dict' entry of Type Secondary. Starting with 'Enter' equivalent to an ASM 'call'.
This becomes a new thread. Sequenced by NEXT, continuously. Until ';' runtime closes it.

Next we will see an evolution... Slow, as usual practice is to base on the 'known'.

Proto-OOP

Each word is a of a type. Either code or data. To run a thread or point a variable.
':' indicates the type "pack of FORTH instructions", compiled there to be dispatched.
Means to build a thread, to execute when invoked. The type (CFA) will managing what follows it.
This to say that a 'thread-type' identifier points code that will deal with a new sequence.

A thread needs its own IP to Sequence its own commands list. Nothing weird, nor difficult, similar to what a CALL in assembler would do. Difference being, you do not say "CALL <word>".
You say "<word>", and its type Enters it. It's the equivalent of a CALL, saving its place for a Return.
(New 'words', routines freshly built, are referenced as CPU instructions would.

Example: A simple Word's Body: On an emulation of '2*', after the Enter CFA,
'**DUP**' follows, is dispatched (and IP advances). To note DUP is a primary word, meaning
DUP own Type says: "Code ahead", as code is the pack following the 'type'.
'+' is dispatched the same way (and IP keeps going)... Finaly,
';' is dispatched, it's CFA saying "EXIT", for a return (closing the Do Enter placed by ':')
It pops the previous-thread IP, returning to the sequence. Just as a CPU does.

New words are run, as a CPU. <u>But allowing instruction-set expansion</u>. <u>And </u>self-management.
As long as principles are respected, registers are kept correct, everything runs smoothly.
It's of little importance, HOW it's done on a particular CPU. Only an efficient system.

In short, the 'type' is code, 'words' start with a 'type'! (That works with variables too)
This simplifies a compiler, each item responsible by itself. Lets now picture this:

Anatomy of a Word:
It's an Object of 1 Method, and a sequence of Data. The method is the word TYPE:
Be it a variable, a FORTH word, a constant, or a user defined Type (as an array)
(BTW, building new types is a major quality of FORTH... Thus the DEFINER DOES> pair)

```
    +===========+-------------------------------------
    |  Method  |   DATA   |   DATA   |   etc...  |       ; ! The great unification/simplification !
    +===========+-------------------------------------

The above translates into a Primary Word (Type Action):    ; Were the CFA...
    +===========+-------------------------------------
    |    CFA    |    Z80    |    Z80    |    Z80    |     ; Redirects the dispatcher to the following OPs
    +===========+-------------------------------------

... a secondary Word (Type Forth):                         ; A CFA always points code to deal with a Type
    +===========+-------------------------------------
    |    CFA    |   Forth   |   Forth   |   Forth   |     ; Emulates a call into a Forth Words thread
    +===========+-------------------------------------

... a Konstant, a Variable:                                ; Forth Words are also 'data'
    +===========+-----------
    |    CFA    |   DATA   |                               ; The CFA code satisfies a data type
    +===========+-----------
```

Thus, any user-defined Type can be build and run -- I.E., its associated data (single, or multiple).
(On those Types, the CFA points to the DOES> action -- the user method to deal with what follows)

To be reminded: <u>It's all about self management</u> (20 years before OOP).
- Every Word is a Code Method (type) followed by its own particular Data.
  Be it a user-defined Array... A user-secondary word... or a Forth-primary Word.

Also making easy for the programmer, to extend the Word-set. (Primaries easy with the CODE Word).

Threading

Beyond Type ‘self-awareness’, Words do threading. Let's see how that is done.

FORTH Registers are: ... as a virtual CPU would

```
    1 - IP Instruction pointer to sequence being run (equivalent to CPU Program Counter, PC)
    2 - W Local Work pointer, to ‘word’ being (a code address, equivalent to CPU microcode)
    3 - SP and RSP Stack Pointers, splitting the stack function into Arguments and Threading
    4 - UP optional register serves to point SystemVariables (local to a user or task)
```

<u>Threading means to Enter/Exit</u> ... into/out a word, emulates a call out of the (NEXT) sequencer

```
        Enter:          ; Emulates Asm ‘call’
    1 - Rpush IP        ; Save Instruction Pointer
    2 - IP <- W         ; Local W (ParameterField) is new IP (1st word of the new thread)
    3 - Next            ; ...ReStart dispatching

        Exit:           ; Emulates Asm 'Return'
    1 - Rpop IP         ; Restore Instruction Pointer back
    2 -                 ; This level is absent (no ended)
    3 - Next            ; ...Continue dispatching (the previous level follow-up)
```

<u>Dispatch is done by the NEXT routine:</u>
This is a system routine, assisting words. NEXT sequencer and Enter/Exit threading cooperate automaticaly.
The Original FORTH dispatcher is the Indirect method. Direct method is easier, but mixes code and Data).
Indirect keeps DATA and CODE apart. No mixes. Thus, more general, usable on ANY CPU.

```
        Next:           ; Fetch and Dispatch
    1 - W <- [IP++]     ; Get Instruction trough I-Pointer (counter)
    2 - jump [W++]      ; Indirectly jump to Code pointed by W (work)
```

On uCPUs without indirect jumps, as the Z80 is, an intermediary step to get Code is needed.
To mention, that unlike IP, W is temporary (pointing work code address). W++ is may be useful,
as it points whatever follows (ex: a constant). On user definitions, that would be user code.
... Or it may be a data value. On primaries, W++ is useless (and jump [W] is more efficient).

<u>Notes:</u> W first points a code for a Data Type or a FORTH word type. After the ‘++’ it has the ParamField
This brings us to the structure of the Body of a FORTH word:
  * 1st-slot is a CodeField (on W) dealing with what follows. Managing the word Type:
      A constant, a variable, a Forth-Pack-Of-Instructions, whatever type it may be.
  * After and following (as ++W) is the ParamField of that type.
      Meaning everything is data of a Type, except the previous CodeField slot (TypeAddress)

For this arrangement, Forth is proto-OOP. Specially when the language later allowed the user to
build his own types (the DOES> instruction), and even his own compiler words (easier on the ACE).
Proto-OOP again, because each type represent a single OOP-method. (Full static-OOP, using VOCs)
Curious thought, isn't it? Static OOP available without a limiting grammar (scary thought <G>)

A picture values a thousand Words. In the Next section you will find and graphical image
of all this. Bear in mind the structure may change for different implementation.
What matters, is that chosen process of dispatching be correct.

We will examine 2 methods of implementation:
- "Indirect" = which is universal (works with any CPU architecture)
- “Direct”   = Code(executable) and Data(readable) allowed to mix.

___
*# To do more, a Ligh Muse whispered: **“In a poem, the Spear is mighty”**... And Moore shaked "Eureka!".*

As Pictured

Original FORTH, aka Indirect... Sequencing (Next) and Threading (Enter/Exit)

```
Present Word Thread (First Step of NEXT routine)
                      IP   ...  IP+2(on Stk)
                      |          |
                      v          v
---------------------------------------------------------------------
    |   Prev.   |    Word   | Following |          |          |
---------------------------------------------------------------------
                     [IP]
                      =
                      W  ... W+2
                      |
                      v
                  Present Code Word (Primary)  or  Forth Word (Secondary)

If it is a Code Word = Second Step of NEXT routine, the RUN action (INDIRECT):
                      W         W+2
                      |      (not used)
                      v
                  Code Word  (called a Primary Word)
                +===========+-----------------------------------------
                |    CFA    |    z80    |    z80    |    ...    |
                +===========+-----------------------------------------
                     [W]         ^
                      =          |
                      Will execute X code (Remind NEXT runs code, disregards W+2)

If it is Secondary Word = Same Second step of NEXT routine (a word being used code, or Data!):
                      W          W+2 (will be the new IP)
                      |           |
                      v           v
                +===========+-----------------------------------------
                |    CFA    |   Forth   |   Forth   |   Forth   |
                +===========+-----------------------------------------
                     [W]         IP         IP+2
                      =
                      Will execute X TYPE code: initializing a new thread |or| processing data
                        docolon does Enter: Push Old IP+2, IP gets W+2  || dovar places W+2 on Dstk
```

The last step being the RUN step, It is a<u>lways a Jump to CFA asm code</u> (Type or Primitive).
All that asm code ends running NEXT (curious name), to dispatch the next Instruction (as a CPU).
Each New Thread branch emulates a ‘call’ as a CPU would. The difference? It is a Type, not an OP.

A Few Comments:
Naturally, if an instruction starts with COLON, an Enter will be executed for that word.
Then going inside it, dispatching by their type (Note: There's one CFA, Exit is called by ‘;’)

Again: _NEXT & _RUN code make the Dispatcher. _ENTER and _EXIT expand threading to another level.
_EXEC is the least important, it serves a FORTH Word with a similar name (get addr and _RUN it)
Is a system routine due its level of action. And close due the use of _RUN (when not a macro)

Remember:
A sub-word definition or CODE-field (in the Header) <u>always points to code</u>.
... This code deals with the DATA-field at the word's body, after CFA (type)
... Data may be a FORTH, ASSEMBLER or DEFINED word... Tricky?!? It is easy when you forget
the explanations above, after digesting them. New is hard, forget allows a more complete restart.

What about DOES> ?!?

Simple, it is... But Hell is in details.
Each 'mode' can have several ways to do it. CPU Architectures also have a word to say.
Reason for difficulties here, is overlooking the principles at work.

CREATE-DOES usage is fairly simple:
A definer word prepares, creates a the new named Item into the Dictionary...
One with a precise CFA: One executing the DOES> section defined by the programmer.
Forth offers/delivers a Compiler Word, user defines a Running CFA. An extension of the mighty CFA.

DOES> mounts a CFA on item or structure, allowing the programmer to extend the Forth compiler with new Data Structures (not just composition with available). allowing to define whatever new Type variables (not just adding-up different types). And also allowing to build Control-Structures not available.
Then extending the compiler itself, though not as transparent nor easy (except on ACE-Forth).

This is what DOES> must do (for a newly built Item):
* Link to the DOES> sections actions Address, or to code Entering it.
* Leave the Item Address on stack (so those actions may find the data).

Main thing to retain is: There are two action times... Compiling and Running!
Unfortunately, most examples are variants optimized to Von-Neumann architectures, hiding the methodology behind a shield of complexity (and amazement). These optimizations are legitimate as optimizations, BUT due the amazement ingredient are poor examples. *Things are simpler, close to its principles.*

To build the DOES> extension system, no matter the Sequencer method, forget particular recipes.
Then focus on the goals! Grasp them! Follow what I call the Newton method: DIVE into them.
Always remind Forth words start with a Running TYPE (CFA) to deal with what follows.

On the original and universal Forth, DOES> principles are fairly simple:
A definer word creates a the new entry on the Dictionary... with an added run section defined by the programmer. The easy to use DOES> word marks an item-actions section.
Because far away, this demands a different (Enter), the (DoDoes) inner routine.
It also needs to place the caller's address on stack, to locate its Parameters-Field.
And then, only then, run the run-part of its class (or type) pointed through the CFA.
What is important, is not to be obfuscated by optimizations (these may come later).

The 'lost' secret: An item is built with a CFA (DoDoes). And then, a 2nd cell: A pointer to the actions in DOES> area. *(Variation is the Item's CFAs to point a bogus CodeField placed before those actions)*

Similarly, DOES> may be replaced by two cells. A (DOES) compiling a pointer on the item, and exiting.
And an (Enter) to the user type actions... *This can be optimized later, then becoming harder to grasp.*
For Von-Neumann CPUs, optimizations replace (DOES) with “call DoDoes”. Make the item CFA to point there without using 2nd Cell Type pointer. Seems cleaner, it just seems so.
DoDoes routine is then different, also doing the Enter (two in one is faster).

Note that original Forth could have not just one DOES>, but several methods. It's OPEN.
Single address, single method, simplifies the user-programmer task. Enough, more efficient.
Indirection shares code for a Type: A common self-managing CFA... It's Simple, it's Brilliant.

On Forth, OOP does not need complex constructions... Unless we need dynamic OOP (usually to avoid).
We already have the essence of static OOP. The remaining is inheritance, though Open-Search Vocs.
And dynamic OOP? That builds temporary instances, demands RAM management (aka garbage control)
If needed, it can also be built... with many solutions to be considered, not just copied.

On tech, to copy is always cheap, inefficient, not engineering. While keeping all simple does pay.
Specially when it forces the user-programmer to get a better sigh on the problem to be solved.
Note: Check the "1% Code" article from C.Moore, available on his Color Forth pages.

What makes a 'Forth' ?

The stack is just a detail
For many, Forth is seen covered by the usual sight of the Stack.
View enforced by the use of two stacks when primitive CPUs delivered just one.
Only to be reminded that is just a tool for managing arguments and temporary results.
Then reminded the two stacks is a simplification to ease programming, reflected as efficiency.

To most, Forth relates to 'threading', not knowing what threading means in the computing context.
Or the "reverse notation", a fancy way to say arguments must be available BEFORE acting on them.
Such being just a direct way to communicate without the overload of a syntax (an apparent exception are instructions were the argument MUST NOT BE evaluated by the commands interface, aka interpreter.

===
There's Type, but there's Build Types
We mentioned the main element of Forth to be the the CFA (type) followed by a list of that data type.
The DOES> mechanism builds the final attribute of Forth: Compiler level Type generation... at hand.
These two elements represent two evolutionary steps made over LISP suggestions and experience.
The third element being the double Stack simplification (something LISP should had adopted).

Forth CFA vs Code (in place of CFA)
We have mentioned Primitives CAN be optimized as directly pointed as a CF (on 'Direct' and SBR).
We also need to answer a question: Is a SBR implementation... still ‘a’ Forth?
The xt (eXecutable Token) is replaced by { CALL xt]... that's irrelevant.
If some xt's are optimised towards native... it's still irrelevant.

It's the capability of a user, to build proto-OOP NEW types what makes FORTH. NOT unusual details.
And that's what DOES> does, whatever that may do: Be it a new Data type, or a new Functionality.
As long as DOES> is available to the user/programmer, benefits justify the engineering effort.
The might of the CFA is not disturbed, by an extension of his own image. It is a bonus.

Forth ‘Easy’ Disassembly (but some inner routines)
Another point to consider, not 'Forth' but a benefit traditional ‘allowed’, is easy disassembly.
ACE-Forth (again) added a few strategies to reconstruct a SOURCE, thus keeping Structures Error-Checking.

Can SBR do the same?!? Hardly. Greater compromises are needed, specially if optimization towards native code is desired. But yes, it can be done. What will happen then?!? (CFA in Words are a proven concept.)

===
Were to go from Forth (?!?)
The question of "what then" is a valid one, when rewards are on hardware. Not exactly the point.
The problem is the usurpation of benefits. By ‘copy’ or by 'teaching, for profit (monetary or otherwise).
Every effort rewards hardware makers. As things are, ingenuity seems a loss of time. To copy, is favoured.

This is a problem never solved. The usurpation of other people work. It's surpassed by plagiarism in Physics, as trashing it on Astro-Physics evidences. It could be worse... It could be Pharmaceutics trashing Medicine to oblivia, then causing too much suffering. (Would fail if omitted the following Off-topic:) Now supporting the extermination of Mankind while building a ‘master-race’ on hypothetical grounds (also delivering different poisons to different 'targets'). The whole disguised under quests and by arguments played. (The most strange suggestion of to re-build Jesus was an helping disguise.)

"What makes a Forth" is a (much more) pleasant question.
Answers are plain, thus not deniable. Without chocking we could only ask "What makes a Buddha?".
The meaning being “What makes a Human?” . Hint: My dog was a real person, also a real Scientist.

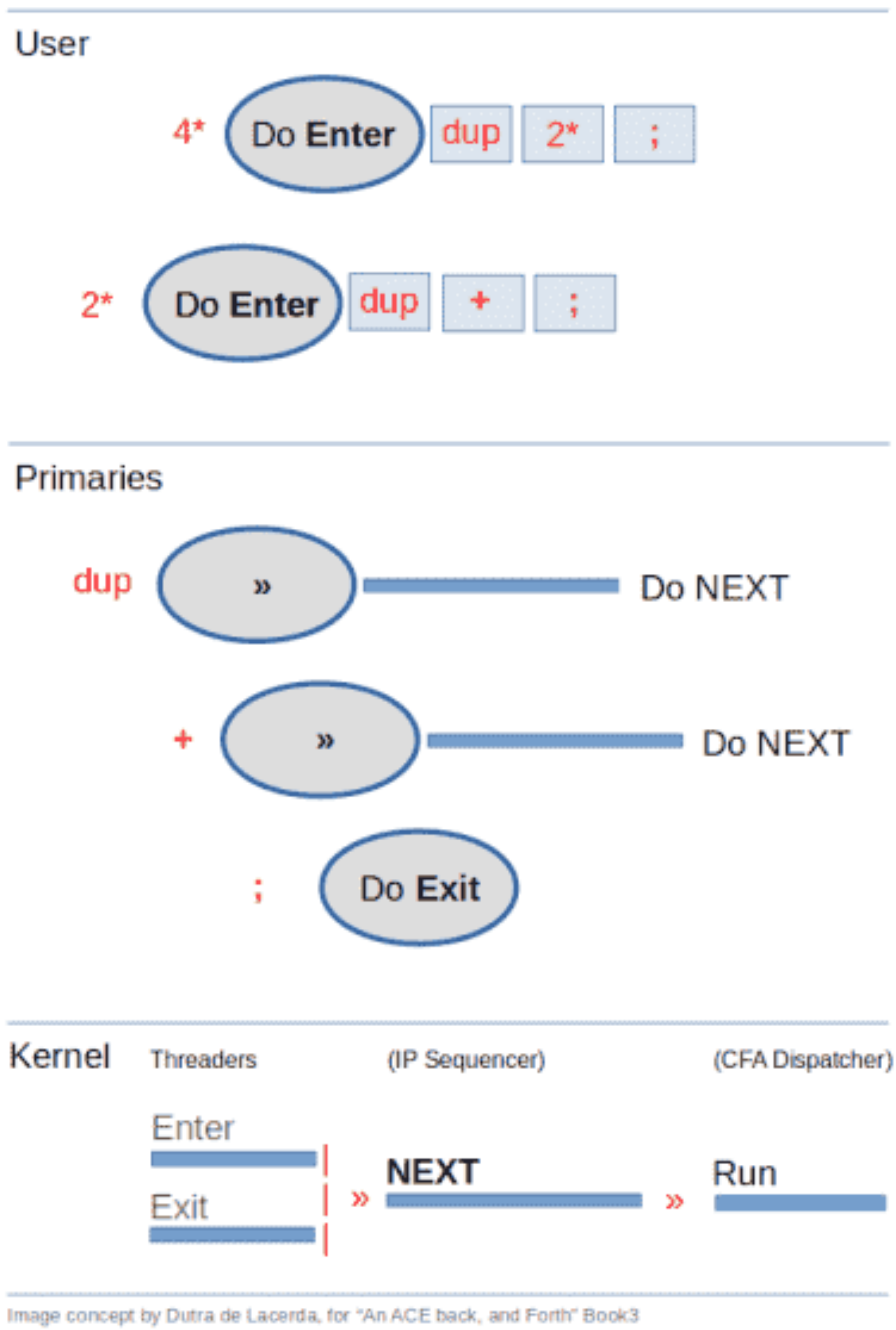
===
And Knowledge? Its overrated (effect of 'zoocial play"). Better work it well (read ingenuity).
"Status plays", are a bit like pressure cardboard, painted to fake 'wooden' toilet seats.
(Check "Grains of Salt" in Appendix: The hidden in plain sigh, to recognize the call.)

•

Chapter 2 - Dispatcher 'modes'

First, in preparation... A picture to remember later.
Never made, seen or told, it's a life-jacket (or a time-saver)
Start give it a quick look... Forget it... Then come back often.
(Do not brag the shared as own, nor use it to impress the weak. Earn it!)

Hard *Question: Why never shown in so many decades? Maybe because the HOW is very rarely, a lost WHY. Equivocations are a profitable practice*. While you can, Help children grow (neither slaves, nor tyrants).



A) Indirect method (universal)

InDirect Method is the original one.

Forth Registers

As mentioned, the Forth Program Counter is called IP (Instruction pointer).
It is a pointer to a word being dispatched: Be it a Forth primitive, be it a type (a data object method). Note: A Forth Secondary is also a type (a sequence of words).

The second pointer, W, is a work pointer to the code to run. (see pseudo-code bellow)
I.E. the content is the WORD-TYPE, named CodeField-Address (a pointer to TYPE code).
On the Z80, to jump indirectly to that code (to RUN it) we need to use a temp register.

WHY no doing that directly? Secondary words are seen as DATA, not as code.
It's coherent, it's universal.

Reminder

```
 NEXT: W <- [IP++]     ; Get Instruction trough I-Pointer (caller)
  RUN: X <- [W++]      ; Get Code Address trough W-Pointer (Work)
       jp   X          ; Run X -- THE code address, in a Reg.
```

Note that CPUs should allow an indirect jump (trough a pointer, an address).
as jp [W], being desirable a jp [W++]. Some as the 6809, did. The Z80 did not.
Thus the use of an intermediary Reg X, to receive the code address to run into.
Warning on Z80 terminology: JP X is JP (HL), meaning the address content of Reg HL.

Also remind, that code serving FORTH dispatcher (a virtual CPU), will return from execution to the dispatcher. Simply by jumping back to the dispatcher.
'Returning' to the Next routine (as a CPU dispatcher would, unnoticed).

Two main Z80 solutions... and a small optimization:

```
(1) Z80 wisdom Programming tells to keep Registers available:
        IP is kept on CPU-Stack, or in a RAM location.
        On CPuStk, Next is faster.by 11T (21T vs 32T)
        This was the initial focus of Forth implementations
        Also, considers D-Stk a foreign stack, thus an added Soft-Stack
        Therefore FORTH has it's own Data-Stack (this was later found to be slower)

IFF Z80 ASM wisdom is discarded, ie, CpuStk is used as D-Stk, that situation inverted
Then, the 11T loss is covered by a faster D-Stk. This brings up a new situation

(2) Counter-intuitively keep IP on a Z80 reg (1 reg less for ASM work)
        Using CPU-Stk as D-Stk makes it faster, R-Stk is now the ‘foreign’ Stack
        This allows to sacrifice a Register, saved elsewhere if needed
        As R-stk access is less frequent, slowing it down has little effect
    This has two consequences:
        D-stack is considerably faster.
        IP usage is immediate, saving 21T
    Result is this scheme (IP on Reg and CPu-Stk) near doubles Soft-stk speed

(3) As above, with a small detail can be added:
        IP still needs to be increment in 2 steps (inc IP, Inc IP)
        But W register is rarely used by primitives. Thus delaying W++ on a
        “only if needed” basis... will save 6 clocks on NEXT on most Primitives.
    This is a small optimization, but it pays on many slightly faster operations.

Note: Every instruction is a pointer to code. It is ALWAYS an address.
(It's Data. This makes it universal, Harvard CPUs compatible.)
```

Z80 wisdom (the ACE)

```
// Traditional 8080/Z80 implementation. All Regs available.
//
// IP = Instruction Pointer  ;Stack saved = Next word (in the caller).
//  W = Working Pointer      ;DE scratch  = Current word, or body of code.
//  X = Auxiliary Pointer    ;HL scratch  = JumpToCode (pointed by W)
//
// Push, Pop and JMP are native assembler instructions
//  2  = is WordSize (16bits) in bytes for 8 bit CPUs like the Z80
// CPUstk is RetStk --- IP is Saved on CPU stack --- W is temporarily on DE
//=======================================================
// Inner Loop = Executing a Forth Word in a row
_Next:
        IP := Rpop      // Restore IP (from CPUstk)
_Nsub:  W  := [IP++]    // W gets next word CodeField Address (while IP advances)
        RPush(IP)       // Save IP (to CPUstk)
_Run:                   // After RUN, NEXT totals 85 Clocks (due CPU Pop and Push)
        X  := [W++]     // Get Link Value (type or code), W ends pointing ParamField
        JMP (X)         // on Secondaries it's TYPE, on Primaries it's ACTION
//=======================================================
// Enter inside a word (NeSting = Going Deeper)
_Enter:                 // This is TYPE ForthWord (start new thread)
                        // Old level IP already on stack
        IP := W         // W is still on DE, thus its a fast (EX DE,HL)
        JMP _NSub       // It's _Next without the Pop(IP)
//-------------------------------------------------------
// End of Word = Return Back (Un-NeSt = Return)
_Exit:
        IP := RPop      // Drop this IP, _Next will get the old one back.
        JMP _NEXT       // continue at this level
//-------------------------------------------------------
// Execute a Forth Word (address on the Forth DataStack)
_Exec:                  // Get Addr from Top of Data as Next would
        W := Dpop       // Just Run it... Its CFA will take care of its Type
        JMP _RUN        // ... before running it
//-------------------------------------------------------
// No more Dispatcher <G>
```

Result example

```
  DUP:
   (run)    ; CFA Run (2nd stage of NEXT)
   Dpop DE  ; SoftStk is slow, 58 clocks when inline
   Dpux DE  ; SoftStk is slow, 89 using present RSTs
   Dpux DE  ; ... Got TOS on DE, duplicate it
   (next)   ; +8+85 clocks, will continue with (run) ...another word
```

Actual code

```
  DUP :         ;--; label of code address
      RST 18h   ;89; Dpop to DE from DataStk
      RST 10h   ;89; Dpush DE   into DataStk (restore Top Value)
      RST 10h   ;89; Dpush DE   into DataStk (duplicate it)
      jp(iy)    ;93; Jump NEXT = Proceed phase, back to the Dispatcher
;------------- ;=360
```

___
“Easy peasy” as the French say. Similar to English but with a different accent.
(They do, it's the accent what may confuse you.)

Forget the Z80, go Forth

```
// New 8080/Z80 implementation. One main Z80 Reg lost (now IP)
//
// IP = Instruction Pointer  ;BC stored  = Next word (in the caller).
//  W = Working Pointer      ;DE scratch = Current word, or body of code.
//  X = Auxiliary Pointer    ;HL scratch = JumpToCode (pointed by W)
//
// Push, Pop and JMP are native assembler instructions
//  2  = is WordSize (16bits) in bytes for 8 bit CPUs like the Z80
// CPUstk is DataStk --- IP is Saved on BC (as lesser reg loss)
//====================================================
// Inner Loop = Executing a Forth Word in a row
_Next:
                        // IP on available Z80 reg (BC)
        W  := [IP++]    // W gets next word CodeField Address (while IP advances)
                        // To Run word, get Code from CodeFieldAdress
_Run:                   // After RUN, NEXT totals 68 Clocks (No PushPops)
        X  := [W++]     // X points CodeAddress; W becomes ParamField Addr
        JMP (X)         // on Secondaries it's TYPE, on Primaries it's ACTION
//====================================================
// Enter inside a word (NeSting = Going Deeper)
_Enter:
        RPush(IP)       // save the caller (on the much slower Soft-Stack)
        IP := W         // set's IP as that deeper word
        JMP _NEXT       // into the dispatcher
//----------------------------------------------------
// End of Word = Return Back (Un-NeSt = Return)
_Exit:
        IP := RPop      // get back previous upper thread
        JMP _NEXT       // into the dispatcher
//----------------------------------------------------
// Execute a Forth Word (address on the Forth DataStack)
_Exec:                  // Get Addr from Top of Data as Next would
        W := Dpop       // Just Run it... Its CFA will take care of its Type
        JMP _RUN        // later back to the dispatcher
//----------------------------------------------------
// No more Dispatcher <G>
```

Result example

```
DUP:
   (run)    ;38; CFA Run (2nd stage of NEXT)
   Dpop DE  ;10; CpuStk is fast, 10 clocks
   Dpux DE  ;11; CPuStk is fast, 11 clocks
   Dpux DE  ;11; CPuStk is fast, 11 clocks
   (next)   ;42; will continue with the (run) of another word
```

Actual code

```
  DUP :        ;--; label of code address
      Pop  DE  ;10; Dpop to DE from DataStk .. ; get
      Push DE  ;11; Dpush DE   into DataStk .. ; restore
      Push DE  ;11; Dpush DE   into DataStk .. ; duplicate
      jp(iy)   ;76; Jump NEXT = Proceed phase, back to the Dispatcher
;------------- ;=108
```

___
“Easy peasy” as the German say. Similar to English but with a different accent.
(They do. Again, it's the accent what may confuse you.)

A bit less, for a little more

```
// Later 8080/Z80 optimisation, by not actualizing 'W' (do it later, if needed)
//
// IP = Instruction Pointer  ;BC stored  = Next word (in the caller).
//  W = Working Pointer      ;DE scratch = Current word, or body of code.
//  X = Auxiliary Pointer    ;HL scratch = JumpToCode (pointed by W)
//
// Push, Pop and JMP are native assembler instructions
//  2  = is WordSize (16bits) in bytes for 8 bit CPUs like the Z80
// CPUstk is DataStk --- IP is Saved on BC (as lesser reg loss)
//===================================================
// Inner Loop = Executing a Forth Word in a row
_Next:
                        // IP on available Z80 Reg
        W  := [IP++]    // W gets next word in this level
                        // To Run word, get Code from CodeFieldAdress
_Run:                   // 6 clocks faster = 62 Clocks (W actualization delayed)
        X  := [W+]      // X =CodeAddress; W turns W+1, not yet ParamAddress
        JMP (X)         // on Secondaries it's TYPE, on Primaries it's ACTION
//===================================================
// Enter inside a word (NeSting = Going Deeper)
_Enter:
        R-Push(IP)      // save the caller before going deeper
        IP := W+1       // correct W to finally points to ParamAddress of the word (new level)
        (copy of _Next) // run the dispatcher (code inline not to Jump to it)
//---------------------------------------------------
// End of Word = Return Back (Un-NeSt = Return)
_Exit:
        IP := RPop      // get back previous upper level
        (copy of _Next) // run the dispatcher (code repeated avoids Jump to it)
//---------------------------------------------------
// Execute a Forth Word (address on the Forth DataStack)
_Exec:                  // Get Addr from Top of Data as Next would
        IP := Dpop      // Just Run it... Its CFA will take care of its Type
        (copy of _Run)  // CodeField, later back to the dispatcher
//---------------------------------------------------
// No more Dispatcher <G>
```

Result example

```
DUP:
   (run)    ; CFA Run (2nd stage of NEXT now 8 clocks)
   Dpop DE  ; CpuStk is fast, 10 clocks
   Dpux DE  ; CPuStk is fast, 11 clocks
   Dpux DE  ; CPuStk is fast, 11 clocks
   (next)   ; 32 clocks (will continue with _run ... on another word.
```

Actual code

```
  DUP :        ;--; label of code address
      Pop  DE  ;10; Dpop to DE from DataStk .. ; get
      Push DE  ;11; Dpush DE   into DataStk .. ; restore
      Push DE  ;11; Dpush DE   into DataStk .. ; duplicate
      jp(iy)   ;70; Jump NEXT = Proceed phase, back to the Dispatcher
;------------- ;=102
```

___
“Easy peasy” as the Russians say. Similar to English but with a different accent.
(They do, it's the accent what may confuse you. The meaning is the same.)

A short-cut: The Direct variant

The 'DIRECT Method is (just) a variation... a small shortcut when CPUs allow mixing code and data.
Instead of CFA with an Address to machine code, it already contains a fast JUMP <code address>.
Or a CALL, to get the parameter field address, if needed (a POP can be faster than W+Cell).
Or the Primary that would follow its CFA. This speeds up primary words, the most used.

This <u>cannot</u> be used on Harvard CPU architectures... Nor is advisable on Segmented Programming.
Segments separate code from data. On the x86 these, can be the same (then limited to 64K +SS).
Indirect allows 64K +StackSegment +n*64K (these, multiple user-segments and library segments).

```
NEXT: W <- [IP++]     ; Get Instruction trough I-Pointer (caller)
 RUN: jp   W          ; Run X -- To a code Address (in user data space)
```

Note that this JUMP, being direct, shortens NEXT (primitives benefit, plenty).
That is of little help for Secondaries. But most work is done by Primaries code!
<u>*Warning on Z80 terminology:*</u> *"JP X" was Z80 OP JP (HL), to Reg HL content* (direct jump).

In short, every word Code Field (Word Type) become ASM to run. Primaries no longer having a CFA.
Next is faster, but for them only: The 2nd indirection equivalent is on the lesser run ENTER.

As in the Indirect method, the Forth Program Counter is IP (Instruction pointer).
This is a pointer to a word being dispatched: The CODE of a Forth primitive, or of Type
(a data object method). Note: A Forth Secondary is also a type, a sequence of words (its addresses).

The second pointer, W, is now has a code address to run.. directly.
Not to a pointer address. I.E. no longer the content of the CodeFieldAddress, but a CodeField.
A very short Code, usually a JUMP, a CALL as when W was needed to get the BODY Address.

Remind this: Only possible on CPUs where DATA and code can be mixed.
Not universal, but faster. And X no longer needed. Also easier to grasp.

DIRECT vs Indirect: Direct mode is <u>not</u> Universal. It's just an optimized variation.
Words do not start with a CodeFieldAdress slot (cell). Start with Asm in Data space.
Violate any Code/Data segmentation... specially on CPUs with an Harvard Architecture.
Where used, <u>it benefits Primitives</u> (where most workload is)

For the Z80, by the 90's, DIRECT was a faster solution, 40% faster than the best Indirect.
DIRECT conjugates Speed and Size with ALL Forth characteristics, as user defined types,
aka DOES> words. It's also simpler. Faster would be the slightly bigger SBR method.
(Only Indirect method being both simple and universal... Ingenuity is needed.)

Question: "What-If" it is a non-Primitive? Either Data or Secondary?
CodeField (no longer CFA) needs a short code, a JUMP or CALL <Address> replacing indirection.
Direct Dispatching ‘just’ benefits Primitives, yes. But that's where speed lies (~35% faster).
This also frees a CPU reg, then allowing DataStack optimization: Top-of-Stack on reg,
TOS on reg further speed it up by maybe ~10% (average). For a total of 40% speed-up.
Now measured with Weights..No longer by Sieve, nor by testimonies.

Warning: Speed-Ups mentioned... are for Z80 alone (and only by the 90's)

<u>Developers Note</u>:
If using CPUstack, as should, using a CALL disturbs the Stack, needing correction.
((It compensates with a benefit: The ParamAddress is pushed into the CPUstk.
((Thus not needing to calculate W++ with the size of CALL <Address> slot.
For an ENTER, that means doing >R (the action, not the word).
For a DOES> that it is a bonus, allowing a quicker action.

CodeField is now ASM

```
// Experimental by 1982(?). At least one Reg lost with IP (BC)
//
// IP = Instruction Pointer  ;BC stored  = Next word (in the caller).
//  W = Working Pointer      ;HL scratch = Only IP is kept, W is a scratch Reg
//  X = Not used, it's free  ;DE is free for work with HL
//
// Push, Pop and JMP are native assembler instructions
//  2  = is WordSize (16bits) in bytes for 8 bit CPUs like the Z80
// CPUstk is DataStk --- IP is Saved on a Reg (a reg loss)
//--------------------------------------------------------
// Inner Loop = Executing a Forth Word in a row
_Next:
                        // W gets this level and goes deeper making it current
        W := [IP++]     // get next IP in row (this level)
                        // To Run word, just run the word CodeField
_Run:                   // RUN Primitives = 36 Clocks (Run Call-CFAs = 36+17+10=53)
                        // W points CodeAddress; ParamField follows CodeAddress
        JMP  W          // execute THIS Code pointed by W (TYPE of word, or code)
//--------------------------------------------------------
// Enter inside a word (NeSting = Going Deeper)
_Enter:                 // On the Z80, using Call on CodeField calculates ParamField
        RPush(IP)       // Save this level IP
        IP:= Dpop       // Get ParamField already on CPUstk (due the call)
        (copy of _Next) // run the dispatcher into the new loop
//--------------------------------------------------------
// Exit from Word = Return Back (Un-NeSt = Return)
_Exit:
        IP := RPop      // go back to caller word
        (copy of _Next) // run the dispatcher, into the new loop
//--------------------------------------------------------
// Execute a Forth Word (address on the Forth DataStack)
_Exec:                  // Get Addr from Top of Data as Next would
        W := Dpop       // Just Run it... Its CFA will take care of its Type
        JMP  W          // (copy of _RUN) executes the CodeField pointed by W
//--------------------------------------------------------
// No more Dispatcher <G>
```

Result example

```
DUP:
   (run)    ; CF Run is absent ( _Run is now only 4 Ts)
   Dpop TOS ; CpuStk is fast, 10 clocks
   Dpux TOS ; CPuStk is fast, 11 clocks
   Dpux TOS ; CPuStk is fast, 11 clocks
   (next)   ; This will continue with _run on another word.
```

_Next and _Run now is #6 bytes, and _run is #1 byte. These #7 bytes of _Next and _Run can be a macro: code placed inline on crucial OPs saving a JMP, Then saving 8 clocks on selected OPs. (Notice: We also spend 21 clocks to fetch TOS. That will lead us to another optimization.)

Actual code

```
  DUP :       ;--; label of code address
      Pop  DE  ;10; Dpop to DE from DataStk .. ; get
      Push DE  ;11; Dpush DE   into DataStk .. ; restore
      Push DE  ;11; Dpush DE   into DataStk .. ; duplicate
      NEXT     ;38; NEXT inline (because a critical word)
;------------ ;=70 Clocks
```

Fast fast fast, implies no safe mode.

___
 # Cracking eggs gives us a great, healthy meal. After crhacked.

CodeField is ASM + TOS cached

```
// Experimental by 1982(?). Two Reg now lost: One for IP, another for TOS
//
// IP = Instruction Pointer  ;BC stored  = Next word (in the caller)
//  W = Working Pointer      ;HL scratch = We loose 2 Regs, HL is always available
// TOS= TopOfStk             ;DE stored for a faster access. Always needed, rarely a delay
//
// Push, Pop and JMP are native assembler instructions
//  2  = is WordSize (16bits) in bytes for 8 bit CPUs like the Z80
// CPUstk is DataStk --- IP is Saved on a Reg (a reg loss)
//---------------------------------------------------
// Inner Loop = Executing a Forth Word in a row
_Next:
                        // W gets this level and goes deeper making it current
        W := [IP++]     // Get next IP in row (this level).
                        // To Run word, just run the word CodeField
_Run:                   // RUN Primitives = 36 Clocks (Run Call-CFAs = 36+17+10=53)
                        // W points CodeAddress; W will be ParamField Addr
        JMP  W          // execute THIS Code pointed by W (TYPE of word, or code)
//---------------------------------------------------
// Enter inside a word (NeSting = Going Deeper)
_Enter:                 // On the Z80, using Call on CodeField calculates ParamField
        R-Push(IP)      // Save this level IP
        IP := TOS       // Get ParamField already on CPUstk (due the call)
        TOS := Dpop     // Actualize TOS reg (an extra Operation where costs little)
        (copy of _Next) // run the dispatcher(a copy of Next+Run) for the new loop
//---------------------------------------------------
// Exit from Word = Return Back (Un-NeSt = Return)
_Exit:
        R-Pop(IP)       // go back
        (copy of _Next) // run the dispatcher(a copy of Next+Run) for the new loop
//---------------------------------------------------
// Execute a Forth Word (address on the Forth DataStack)
_Exec:                  // Get Addr from TopOfData as Next would
        W := TOS        // Just Run it... Its CFA will take care of its Type
        TOS := Dpop     // Actualize TOS reg (an extra Operation where it costs little)
        JMP  W          // (copy of _RUN) executes the CodeField pointed by W
//---------------------------------------------------
// No more Dispatcher <G>
```

Result example

```
DUP:
   (run)    ; CF Run (2nd stage of NEXT) Now only 4 clocks
;  Dpop TOS ; No Dpop if on Register. Saves 10 clocks
   Dpux TOS ; CPuStk is fast, 11 clocks
   (next)   ; This will continue with (run) another word.
```

NEXT (next)+(run) could be a macro, saving 8 clocks for 7 bytes. Question is...
What is the real balance of that exchange? We did maybe better, with having TOS cached on a Reg:

Actual code

```
  DUP :       ;--; label of code address
      Push DE  ;11; Dpush DE  into DataStk .. ; duplicate
      jp(iy)   ;46; jump NEXT (not inline, but so frequent inline makes sense = 38 clks)
;-------------;=57 Clocks    (next inline, total would be 49 clocks)
```

After saving so much, why not lose a miserable 8 clocks? For safety?

 # Making scrambled eggs can makes us cry: Such beautiful eggs, now so scrambled.

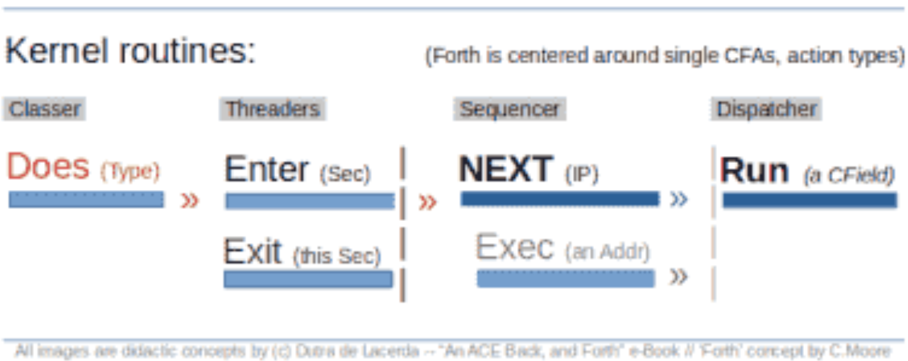
A better Map

Whenever anyone dives into Forth System texts, the mechanics are shown. These are understandable, but also limited to a particular implementation. The spirit behind the ‘thing’ is somewhat missed.

Students, voluntary or not, face endless equivocations due a satisfying How, overlooking WHY.
The integration of the parts is simplicity generating complexity. It's easy to BE lost.
(Thus we limited this whole chapter to the NEXT routine, showing it as just a choice.)

In preparation for a sensible whole, some facts rarely mentioned (if ever) should be noticed. These are actually quite simple, hidden on plain sight. Are overlooked, and that for decades. Examining code is useful, but it induces an undue amazement... and confusion. This to say:

The genius of Forth inner working lies on its simplicity and on its integration (then complex). The resulting amazement comes (as usual) from confusion after many particular details. This can better expressed by restoring that simplicity, and its integration:



Simple things are undervalued because simple. Big mistake.
Hard things are preferred, apparently great, and an opportunity.
And yet, only the simple allows complexity. The humble is really glorious.

The picture above rationalizes what is considered as ‘know’... ‘Known’ it is, BUT:
It's not enough, ‘known’ serves only for copies on any theme (no matter its own details).
We see plenty of that on everywhere. Supposedly leading to ‘advances’, most are ‘painting-over’.

A small 'intermezzo'
Be it technology presented as Science, or just ‘zientific’ beliefs solidified on TV or Training, when acceptance is renamed as knowledge, even knowledge supposed understanding, we are all building a prison of dogma... Only because it is desired without notice (for a status).
We must declare to be observed, the failure of ‘Education’ (just training).

Thus the failure of everything ‘believed’ respectable, quickly mimicked for benefits.
We observe the probable reasons, expressed on this parabola of involution:

* First come the wise, establishing centres of progress.
* With them come the close, still sharing its spirit.
* After them the spreaders, much needed.
* Then entropy grows, on the appeals.

As widely ‘know’, dogma replaces the original. Everything ‘continues’ on social games.
Needless to say, this happens at every level, on every theme. We all can observe it daily.
Also on every ‘theme’, by selection (needed, just not when run by desires or by social needs).
... Not because they are bad (seldom are) but because they are misplaced (then subtly imposing).
There are worse things, yes. But these exponentiate every possible distortions.

<u>Thus we conclude:</u> Whatever the ‘theme’, never show everything. Simplify. But not too much.
Not because people do not deserve (they do, eventually)...
But because misplacement is entropic.

Uncomfortable sight? Better than living on a broadcasted ‘reality’...

!!! Recipes are not it

Close to the 90's, Direct+TOS seemed to be the best solution for a Z80 on end-of-life.
But was it the best solution ?!? Different CPU's, different compromises !!!
The various designs available to achieve the same, where:

Addr 'threading' is already covered on DIRECT and INDIRECT sections
    See the previous sections. (Direct can be 10 to 40 %faster than indirect, depending on CPUs)
Times, DUP example: Indirect =360|102 clks... Direct DUP=70|57|49 clks... (On the Z80, ie 8080 line)

SUBR 'threading' (20% to 60% faster than direct method)   ** Measures need to be actualised **
The CPU does the sequencing, even the threading. User code is >50% bigger avoiding the Soft Dispatcher.
    It complicates the implementation of DOES> It's also much harder to isolate the system from user errors.
    Not examined here.
Times Example (not representative), DUP is 37+11 Clocks (avoiding the Soft RStack):
    call xDUP   ; Primitives may use a Reg (or a Ret_Var) instead RetStack ... it's faster
        call=17 + PopRet=10 + (PushTOS=11) + JmpRet=10   \\ Total= 48 clocks (NEXT was 37 Clks)
    This NEXT is different when Ret cannot be HL (SoftStack is the most extreme, it adds 2*75 Clocks)
    It is an interesting exercise on choices: Find new strategies, avoid a SoftStk, still keep simplicity.

NATIVE compiling (Doubles SUBR speed upto 1/3 of HandCoded-ASM)   ** This is an educated-guess **
On most Primaries it reduces 37 Clocks to its SUBR equivalents. May reach 1/2 HandCoded-ASM speed.
    Also not examined here. It builds is CPU code, much-much faster (also on a Z80). CHECK HYBRIDS.
    FORTH versatility is quite reduced. Code-size grows to occupy too much space in 64K boudaries.
Times Example (not representative).   !Notice final code is a mix of Native code and SUBR calls!
    push TOS   ; 11 clocks

In parallel to the above implementations (on the Forth approach) there are other 'variations', even Dictionary 'modifiers', that that can be applied for a particular result: *Either to save space, *either to improve speed *('recipes' are not to be followed strictly, are secondary to goals)*:

TOKEN methods (various speeds, shorter Ops, shorter Secondaries)
There are several solutions to do this... Not examined here either.
((That would lead to a lengthy exposition of many misconceptions. These are a result of cherished classifications, then leading to erroneous beliefs. Lets not fuel useless battles of arguments.))

HYBRID Methods (mostly SBR * Native) We call native when Regs usage is not fixed but table dependent.
It's natural to mix SUBR method with a few FORTH OPs in Native code. (Native is usually Hybrid)
Example: Think DUP not as a "CALL DUP" but as "Push TOS" replacing it in the middle of several "CALL this", "CALL that".
It's also natural to mix DIRECT and INDIRECT (on Harvard uCPUs with fast conditional execution).

Optimizations (Most are particular to each method) -- About native code:
First, Forth OPs can be replaced by similar ASM compound Ops. Secondly, CPU Registers can be invoked by re-assigning CPU registers (as TOS) to Forth registers. Even stack positions, BTW.
    TOS is AX. Later, may be BX, CX or DX. If "where is what" is known, can be adapted on ASM.
Most interesting, always actual, is the basis of any compiler optimization. Forth simplifying it.

In Conclusion:
There are no rules: Goals are the motor, Ingenuity is the fuel. This applies to all designs.
(1) Understanding is just the driver ... (2) The CPU in use what determines what can be done.
(3) In short, it's the designer who decides the balance of choices (by ingenuity, or by a copy).

 # 'Education' is not a vehicle, but a pointer... Attitude IS both.

... Still similar

Recipes are not the real thing... Terms spoken are road signals, not places.

The new kid in town:

A most interesting variant is COLOR FORTH. Follows Forth goals, the CPU revolution, ingenuity.
It uses (and demands) at least 32 bit CPUs. It runs on code Blocks, again an influence from old CPU designs running uCode. Very interesting stuff, full of ingenuity lessons.

Meanwhile, we suggest to solve the previous puzzle. Because, usually, people are teach as if everything is already done, or fully known. All fixed, with no room for inovation.
(NOT to rock a good boat makes sense. To accept and mimic does not.)

> "Not to question, nor to grow, just following the flow"... THAT, is for Dr Dolittle!

Things are hard to find without a light guidance. But already digested solutions are for the lazy.
There may be other perspectives: Find chances to deal with the simple, in a dark (light your light).

> *Why is COLOR so interesting? Why not 'found' before?*
> *What happened, so COLOR FORTH could be 'created'?*
> *Certainly not a reading of the available.*

A bit of untold History:

Soft micro-codding has expressed itself in CPUs, in many ways.
Hard micro-coding has expressed in conditional executing, where a simple flag (in the OP itself) alters the OP meaning. I.E.: The instruction is a block (and new CPUs do that, again).

> On those ways, COLOR FORTH followed the design of FORTH CPUs. So, when translated as virtual CPU on a cheap 32bit PCs (i586), OPs were implemented as 32bit blocks... But COLOR is an Assembler.

Yet, an Assembler where instructions are FORTH (not ANS Forth, naturally) where IP is the CPU PCounter.
As such, COLOR is an Assembler as well as Forth: The result is quite fast, the programs are quite small:

> <u>*Chuck Moore mentioned Color to be (on his hands) "usually 1% of compiled C"*</u>
> *This may sound a surprising exaggeration. Not impossible, Not in the context:*
> *Those 1% meaning a Forth programming approach... not a libraries junk approach!*

Also means the programmer knows the problem, that he will not inject a bulk of 'recipes'.

As C. Moore reminds, from a wide past of experience (on what we call searching the essence):
"To solve 'a' problem, find its particular solution. General solutions are big an inefficient"
Notice similar happens on 'education': The 'student' mounting his library of accepted knowledge.
<u>Most of what is learn are oversimplifications.</u> Some plain wrong, very bureaucratic and mechanical.

Could give many examples... From History, from Medicine. Other 'told' Sciences.
Of many things (here off-topic) 'sold' as right, but forged. Then plain wrong.
... Thus, those 'funny' taglines here placed ending some of these pages.

"FORTH architecture is superb for micro computers.
Many variants should be explored" ~Chuck More, 1989

___
*# Look! There more to see than what the available to you ear*s.
Question! There's much more than you may suppose, Horatio... or any other.

___
A suggestion 2000 years old (at least): *Do not feed wolfs! (wolfs kill the proverbial gold eggs chicken)*

... Just another

Hope the chance given was grabbed, to "walk dark".

On the Harvard Architecture Direct jumps are sill possible, depending on the CPU particular implementation, therefore sometimes faster than the universal INDIRECT approach.

This happens when the equivalent to micro-codding (wide instruction) indicates a conditional (included on the instruction, kind of two in one) or alters the instruction.
Such proto-pipelining allows hybrid dispatchers.

Then, priority is given to the direct method, serving Primaries.
If not a primary, but a word as a Secondary or a Data type, that path is followed.
Resulting in (again) Faster Primaries.

That was the basis of this pseudo-code example:

```
    NEXT:   W := [IP++]                 ; As the CPU PCounter
    RUN :   ?W_condition -> Jump W      ; Direct to Primary
            (else) Jump [W]             ; Indirect to WordType
            ; Note W is not incremented. WordType (CFA) may do that (if needed).
```

Again we remind: ; Also remind our other solution, the ACE++ (not disclosed).
Everything is simple, after told. That is the 'education' delusion.'Solutions' are elusive.
    It's never too much to remind this. (True as this is, we hope the puzzle was faced.)
    True Education is not "I know that", nor "I've heard". That would be gossip.

This solution was not 'found' anywhere, it was engineered.
That is the invitation delivered here: Not to be stuck with recipes.
Also, to focus on the goals. Then, to adapt what is at hand, to achieve them.

*We have detected 12 different(*) implementations, considering that Harvard Architecture doubles the Von Neumann implementation classes. A chart of the main 6 may be in order, to each CPU architecture (2*6 is close enough to both).... Not including the Hybrid modified versions (nor Forth CPUs, adding COLOR Forth).*

___

(*) **Implementation Alternatives according to CPU**:

CPU Architecture is Von Neumann:
- *OPs are either:* \\ these entries are simplified
  - (2) A CodeAddress to execute = indDirectJp/DirectJp (execution by NEXT)
  - (2) An ExecID(token) to follow = table with inDirectJp/DirectJp (by TNEXT)
- *OPs are CPU code:* \\ following options benefit from being mixed.
  - (1) A call (to a routine) = Dispatcher is the CPU, every Forth operator is called
  - (1) A code macro (of code) = Other CPU actions, coexisting with above calls method

CPU follows Harvard Architecture: ... or is a transition CPU, segmented (as the 8086)
The 'perfect' Implementation uses InDirectJmp, CPUs allowing Indirection.
With room to uncommon alternatives: Is DirectJp REALY not possible (there)?
Sometimes there's a way to tweak a method... If possible, is it beneficial?

With any real 'change', compromises to decide may be needed.

___
 # The real questioner motto:
"There are more things in Language and CPUs, Horatio, than are dreamt of by your knowledge."

___
And a warning: *If one copies, honour are due ... Or one it will be stealing, self-inflict a blockage.*

B) Faster SBR 'get' the CPU

The SBR method is an entirely new game: Forth is no longer a Virtual CPU (replacing uCode).
There's only the CPU PC. There's no IP, there's no W. But the lessons taken remain.
IP and W are easy to get, depending on where we placed the Dstk (CPUstk? SoftStk?).

Remind Traditional Sequencer&Dispatcher was:

```
      NEXT: W <- [IP++]     ; Get Instruction trough I-Pointer (caller)
       RUN: jp   W          ; Run X -- Instruction address is not a pointer
```

SBR is no longer coded as sequences of [TokenAddr], but with sequences of "CALL TokenAddr".
CPU Sequencer suggests a similar use, a simple ... CALL <tokenRoutine> ...
That could be enough if Forth was JUST an Interactive MacroAssembler.

Our Forth do longer emulates a CPU. It submits to it. The CPU is now the dispatcher (no adaptation).
IP is now the CPU PC. Every reference is a routine preceded by a CPU call. We loose control of IP.
W is no longer there... Can we restore the functionalities that make Forth beyond the 'looks' ?
  Note: If needed, an IP++ address (PC++) or W++ address can be obtained trough the CPU stack.

We need compromises ... *Will we be satisfied with a fancy travesti of Forth?*
Or Even with slightly faster SBR ? Warning to the honest: Full speed is not easy.
DOES> proto-OOP is absolutely mandatory. But these only seems trivial when solved.
... Go ahead, examine, explore, solve! Do not limit yourself to 'copy' as a study.

It's path of compromises speeding up SBR from 4 times slower than Hand-ASM, to similar speeds.
Or better: Use of work values can be more efficient. And as asier than ASM, more rational.
*Check the C.Moore "**1% Code**" article (in the Appendix). It's not limited to Color-Forth.*

Depending on Choices (as long as not copies) we can reach modern compiled speeds.
These Forth compilers do exist. We salute them... Their 'How' is not to waste:
- The "Why" of choices depend on the ingenuity at work. As things are, it is
  hard to find reasons to share, when the shared is hidden for a contrast.

The "real deal" ... is also build in Forth, naturally.
Over ASM, not to be lost... Just not programmed with edlin.com (not anymore).

A small optimizing compiler is easier to build. Smaller than anything big business 'builds'
exploring public knowledge, solutions by real people, competences paid with public money.
What people do not really own completely (because obtained that way) is then patented
against the original patent rationale. IE, respect for ingenuity, not for fictions.
World progress is stolen in front of our eyes, sovereignty attributed in abusers.

***Patents should be attributed to people**, never to legal-fictions. All leading to*
the "Great Capitalist Soviet", taking the World (under disguises) since 1984.

<< Remaining of SBR sub-Chapter, on efficient compromises, has been removed. >>
<< Know-How make its complexity simple. But personal solutions are private, >>
<< specially when completely original, specially allowing much not yet used. >>

<< Not related: By 2006, we built an universal CRC consuming just 9 clks. >>
<< We noticed it would have military applications, not just Communications >>
<< and Disk storage. Something Corporations supposedly would trade by n10^8 >>
<< Our advise is to erase real advances, not to ease "The Great Distortion" >>

((Internal SBR solutions is know-how to be respected.
((Not to be downgraded, hidden from honest questioners that may or may not find these small texts.
((Pre-SBR descriptions, traditional Forth, are enough to progress on the SBR arena if so desired.

-

Chapter 3 - Going Forward

Forth implementations can be a complex piece of software, of very simple elements integrated together.
This is due its own evolution: Not theoretical but of progressive adaptations, one step after another.
For this integration, we suspect most implementations are based on previous ones giving it a skeleton.

There, L.G.Loeliger's description is can be most useful, though a bit confusing (integration due).
At the end of Chapter 6 of his “Threading Interpretative Languages”, interpreted on interaction.
It lists words by function and utility... classified to ease complexity, by use and nature.

An ASM debug strategy is advised. But also, a Stack Safety mechanisms for the final implementation.
A System that crashes on a mistake, is not pleasant nor efficient. A two systems may be an answer.
More reasonable are earlier decisions, allowing a small loss of efficiency on well placed points.
Development starts with architectural strategies. Sooner or later, experimentation follows:

First, the Dispatcher method is chosen (depending of the CPU), determining the bases:
- The general memory layout and its partitions
- The inner routines... NEXT, RUN, ENTER and EXIT, EXEC   (Dodoes can be added later)
- Hiden primitives ... (branch), 0branch)
- Fundamental CFAs... (konst), (Var), (DoEnter), (DoExit),

At this point, what supports Forth will allow us to follow later our progress:
- I/O routines for char, block, and stream devices.   (used by redirecting words)
- Define Input and Work buffers in the chosen memory layout   (as TIB, PAD, <##>, <BLKS>)
- Number conversion SYSTEM routines.   (as NUMBER?, and WORD?)
- Two sets of internal variables. Shared SysVars and private UserVars multitasking-friendly

Now we can start with the structural elements previously designed on paper:
- The dictionary structure, with its DICT routines, or words.   (DP sysvar, 'CREATE' and ‘,’)
- The DOES> mechanism with the missing DoDoes [SYS] routine
- Atomic-Control-Words are the base for Control-Structures   (JpFwd, JpBck, 0=, 0<>)

Most elements are now in place to complete a 1st Level Core: (still checked with crude dumps)
- A temporary main-loop (to replace later with QUIT as final main-loop)
- Structured Control Words (aka compilers) with their State-based behaviour

Going Hi-Level, exercise the practical and interactive nature of Forth by defining:
- Interface words, to accept user input and words generation.   (as 'QUERY', ‘LINE’, and ‘DEBUG’)

<u>The main interactive loop can be built</u>, the Core can grow   (STATE sysvar, tools)

It's an heavy task because the Core has many interactive elements, joined together.
Hard because parts of the implementation are made in confidence, piece by piece.
All these pieces need to be known, how they interact under our architecture.
Thus the need of several debug strategies starting with ASM... DUMP, DEBUG.
Real implementations are architecturally unique. Only coded Words are not.

No wonder most implementations are copies of lost copies, rebuilt over them.
Yet, no implementation is 'pure'. Not even completely new code...
All have a pedigree of some kind. At least a pattern.

A good source is the very well laid structure of Camel-Forth: Well organized, with a clean skeleton.
That organization can be used as a start for a completely different implementation...
At least deserves to be examined for hints, due its well laid simplifications.

Keep all well organized, for simplicity sake... So to keep the extras coherent.

Real "new code" are innovations, if any. That may come later, after a pattern is followed.
This applies to any language in the planet (batch recipes excluded): To EVERY language.

Fundamental CFAs

Main CFAs are defined immediately after our choice of the sequencer.

On traditional Forth, W points the present Item. It points the CFA, W++ points its ParamField:

```
                      W          W+2 ((on doEnter, W++ will be the new IP))
                      |           |  ((on doKonst, W++ content is to be pushed to Dstk))
                      v           v
                +===========+-------------------------------------
...IND Words... |   [CFA]   |   Cell    |   Cell    |   etc...  |
                +===========+-------------------------------------

[Konst]     ->  Fetch [W+2], Dpush the content fetched ... NEXT

[Var]       ->  Dpush  W  ((it's the item address)) ... NEXT

[Enter]     ->  Push Old IP+2 to return later... New IP gets W+2... NEXT

[Exit]      ->  Pop Old IP ((was IP+2)) ... NEXT

[Exec]      ->  Dpop ... Rpush ... NEXT
```

On SBR Forth, Item CFA is run. There's no IP, nor W ... But the structure is the same. Thus:
(What is our Dstk? For now, we consider it to be SoftStk. Changing it to CpuStk is a good exercise)

```
                  call addr     addr++  ((on doEnter, W++ will be the new IP))
                      |           |     ((on doKonst, W++ content is to be pushed to Dstk))
                      v           v
                +===========+-------------------------------------
...SBR Words... |   [CFA]   | CellField | CellField |   etc... |
                +===========+-------------------------------------

[Konst]     ->  CALL DoKonst    ;; 2nd call, this is W  -- Cell field follows placed on CPUStk
DoKonst     ->  CpuPop Address  ;; Field \ fetch content, Dpush the content fetched, RET to caller

[Var]       ->  CALL DoVar      ;; 2nd call, this is W  -- Cell field follows placed on CPUStk
DoVar       ->  CpuPop Address  ;; Field \ Dpush address  ((it's the item address)) ... RET to caller

[Enter]     ->  CALL DoEnter    ;; 2nd call, this is W -- Cell field follows placed on CPUStk
DoEnter     ->  CpuPop new IP   ;; Rpush AddrJUMP to New Stream of calls \ Each word takes care of itself

[Exit]      ->  RET to caller   ;; to upper stream -- This CFA has no Field following it

[Exec]      ->  Dpop Addr       ;; Jump Addr -- Word will return to caller (Fieldless, as above)
```

Both Methods are not so different, both allow variations depending from Dstk nature (CPU or Soft)
For traditional IND Forth we advise reading “Inside F83” by Dr Ting (he lived up to his name).
An excellent DIR Forth presentation might be “Zen and eForth” by the same author.
From there, an adapted SBR Forth (as 4CMP and Swift Forth) can be built.

Master your Assembler macros facility:
Whatever the case, an Assembler macros file should be included prior to any code or files.
That macros file will grow. Macros will allow code to be shorter. Also reducing errors.
You may want to define your own strategy for debugging... before building utilities.

Know your Assembler macros build system... Be comfortable with it.

Conversion routines

We need to convert a string into a binary value, for storage.
Also need to convert a stored binary value into text.
BASE Variable allows USER to both on any base.

Text-> Number

Conversions to binary are done by **>NUMBER** an elaborate word to fit any value input, and failure to do so.
Its definition was stabilized with FIG and Forth-83 **CONVERT**. Under the argument the name was equivocal.
(note Loeliger's Forth based ZIP code treated text->bin conversion as a single routine).

Historical changes apart, the inner primary (or routine) is exactly the same.
The interpreter (for the OS portion of Forth, not the language portion) has its own testing needs.
These are 'factored' as **DIGIT?** (char, base -- n, flag)... and **NUMBER?** (StrAddr -- n, True | StrAddr, False)
Demanded by the user interface (interpreter), these need not to be fast, are Secondaries.

```
: _DIGIT? ( char -- scale )              ( Convert a character into a scale )
    [ CHAR 0 ] LITERAL -                 ( slide char position to a zero start )
    9 OVER <                             ( ? was char above 9 ? )
    IF 7 - THEN ;                        ( slide down to remove ASCII gap 9..A )

: DIGIT? ( char, base -- value, flag )   ( Try to convert a character into a value )
  >R  _DIGIT?                            ( slide char position to a zero start scale )
  DUP R> U< ;                            ( test scale validity on the base requested )
```

For quick conversion of code data (files) **DIGIT?** should be a primary not to slow down words using it.
NUMBER? can be a Secondary if a small kernel is desired (for migration purposes). At least at first.

Number-> Text

The inner fundamental word is **DIGIT** (value -- char). It begets **EXTRACT** (value, base -- value/base, char)
These should be primaries as they are much used by the system interface (interpreter) even if not so much
by the language (compiled). From there, every following conversion words become trivial.

```
: DIGIT ( value -- char )                ( Convert digit value to a digit character )
  9 OVER <                               ( If value above 9 ->   0 ... IF bellow or equal -> -1 )
  IF 7 + THEN                            ( if value is above 9, enforce A'-'9' gap )
  [ CHAR 0 ] literal + ;                 ( add '0' ASCII offset, to get the final char)

: DIGIT/ ( value, base -- char, quotient )  (Convert value staying ready for the next char )
  SWAP /MOD                              ( reduce value on the base requested )
  SWAP _DIGIT? SWAP;                     ( get char,  zero quotient marks end )
```

DIGIT/ is invoked for unsigned values. Converting signed, the signal must be extracted first
before conversion can start. All then being similar, just different sizes (bytes, integers, doubles).
as later seen on **U. C. W. D**. For now we like to start with **H. HH.** (debug tools built with DIGIT).

For quick conversion to text (lists) **DIGIT** should be a primary not to slow down words using it.
All mentioned particular conversion routines can be Secondaries, while not becoming bottlenecks.

(To be continued)

Console: TIB, PAD, <#>

This sounds simple. Though not related, it must be solved and managed from the start.
Console I/O can be made directly from a device, from BIOS, or a supporting OS.
We need a common common routine to be free from such details. Two are used.

These are (Key) and (Echo). Can be used as vectors to routines (consult Ting, “Inside F83”).
We need at least to get a Keyboard key, and to place a char on Video. Later, with files.
Note all tree devices, Keyb/Screen/File can work at different levels Char/Block/Stream.
(Key) and (Echo) can be called by words at any of those levels. Unifiers can Simplify.

Most common words using these:
(Echo) is used by "EMIT" and “.”
(Key) is used by "KEY" and "KEY?"

We can used Forth without a console, but if interaction is desired, a console will be used.
Then these routines we must have, hidden or ignored as they may be. Excellent solutions.

Buffers are can be tricky. Changes grew from F79 and F83 up to ANS.

I personally feel the ANS document as having mixed natures in conflict.
- *One of unification, not accomplished (a few valid interventions, many demands).*
- *A Bureaucratic presence, mostly felt as an ownership. As a law in the writing.*
- *Good suggestions. Respecting previous forms but lacking structural comparisons.*

It's very easy to agree with C.Moore, when he commented that ANS might have been a mistake.
We enjoy the freedom of Forth, know the need for standards... But we do feel its too much.
Feel that something was lost... We'll start 'simple' with TIB, PAD, and an eye on Files.

In short, it's more a legal document of dubious nature. After it, enthusiasm has fallen.
It become natural for the master designer to abandon the boat, to migrate to other pastures.
... Not allowing the built of ANSI legislation, on his newly created “field of freedom”.

We'll keep closer to F83 (it perfect F79), while keeping the door open for a few ANS suggestions.

Thus, whatever the CPU, computer, or system behind it (if any):

TIB: A Keyboard input buffer is needed. All input words accept it.
It's system buffer. Tasks can have their own not to ever interfere with main one.
Sometimes was set at the end of the Screen buffer, for economic reasons. An hack.
Then used as a source.
It should be independent, have rolling nature (#TIB mod 256)

PAD: A strings work buffer, transient and labile.
It's system buffer. Tasks can have their own not to ever interfere with main one.
It should be independent, have rolling nature (#PAD mod 256)

<#>: Conversion area... An inversion stack (first on top, to be copied straight).
Usually shared with TIB or PAD (but then disabling their desired rolling nature).
It's system buffer. Tasks can have their own not to ever interfere with main one.
It should be independent, fixed and small (64 Chars? Consider binary conversions).

(To be continued)

SysVars and UsrVars

SysVars is a unique global area, destined to the core of Forth.
But when we deal with many tasks, private management are needed: UsrVars.
These manage LocalData, appended to UserVars... This scheme simplifies development.
It also keeps interfacing buffers tidy, even if not including Disk buffers. And design clean.

We may want to keep code on a separate section file, 'included' (called) by our main file.

UsrVars ia a local-task area. Main task has its own UsrVars (appended to SysVars).
As tasks vary, so do Usrvars needs. May have different sized buffers, down to size 0.

As a rule, these buffers are placed at UsrVars end. Priority data first, Options last.

With UsrVars Tasks initialization become simpler, suggesting 3 types: Main, Secondary, Services.
UsrVars and Task private Buffers are initialized on creation (we may latter add other Buffers).
Disk Buffers, when implemented may be shared. Sharing needs to be managed, belong to a service.

Other 'local' Buffers
We already mentioned main Task added Buffers, as TIB, Pad and <#>. These are naturally managed by related vars (in main Task UsrVars). Similar happens with other tasks in a multitask environment.

Each task will have similar ones, smaller than on main task (the main task has wider demands).
Big question is if Keyboard input is a shared service, or not. It will depend from Video input so to avoid surprises, no matter how improbable. Therefore seen as a unity, the "Console".

Inconsistencies on usage must be solved at design time... That will keep usage clean, thus easy.
With that in mind... Main buffers occupy a single area, parted, according variables in Usrvars.
Meaning we allocate Private Areas for each Task, keeping their own buffers separate.

Examples
A general SysVar example, is USER. It's the main reason to split internal Vars. Most are UsrVars.
Note the most important ones are usually cached on CPU registers instead where would belong.

A local UsrVar example, is BASE. It is certainly a UsrVar, for all tasks. Option, (not optional).
Remind on Multi-Task there are 3 types of tasks. They run on different states, BASE to be local.

Concepts must be well digested, not 'eaten'
At this point we need to ask what kind of task EDIT would be... as it interacts with the Console.
If the question is simple, it's also confusing when without a clue. (A copy could be misleading.)
Reason to ask, is that 'knowing' an answer is not to 'understand' a 'given' solution...
It may fail miserably, as Black-Holes or a Big-Bang (both based on presumptions).

Know what you have.
Know were you are heading to... Understand beyond easy 'convictions' (believe nothing).
The above was as easy question. We face bigger questions not made, on real-life, everyday.
Can you imagine the consequences? Of support to equivocations, fallacies, or worse? Reason that.
(Dr. Ting has show to be much above fallacies than the common 'literate'. May 'Forth' helped.)

You may find a few examples (from many more available) on Appendix. This is not an invitation to refuse everything, but to examine everything at hand without pre-judgments, nor pre-concepts...
Trust is a different matter. Whom do you? Should you? Or is it just ...'convenient' (read 'easy')

The moment we write this, Forth is on the frontier of the Solar System, as a uChip, and as code.
We too, Humans to be and others not so much, we are on our own frontiers. We too, are becoming.
All Astro travelers, walking or sailing or navigating on an Astro called Earth, Astronauts.

 # Please do not call that to passengers of little vacuum boats... mere orbito-unauts with big Egos.
Could be worse: Egyptologists. Or Astro-Physicists. Even worse: Medical 'licensees'. ~Dr StrangeHumor

-

Chapter 4 - Runnng Pieces

Service Jumps and Tests

The purpose of structured Programming, is to avoid dangerous Jumps.
These are hidden, used by medium level constructs. Similar applies to Tests.

Are they Boolean checks, or Value comparisons?
There should no place for equivocations. For us, it's practical to allow n = TRUE = Not FALSE.
For comparisons, were equality results on a zero, it's the opposite. (Equal-> Z ... Different->NZ)

Testing Words (and internal routines):

Essential words

0=	(isZero)	This one is crucial. (Zero -> True NotZero -> Zero)
0<	(Signal)	Test signal, the higher bit (signal is also hardware dependent)
U<	(Usmaller)	Unsigned, not an integer
<	(Smaller)	signed

Not essential, can be compounded (slower)

0<>	(NotZero)	[[0= 0=]]
0>	(Positive)	[[0< 0=]]
U>	(Ugreater)	[[SWAP U<]]
>	(Greater)	[[SWAP <]]

Derivate, all can be componded (Much used, near critical. Should be a primary)

=	(EQ)	[[U- 0=]]	... Both signed or unsigned
U=	*(UEQ)*	*[[- 0=]]*	*... The usual alternative*
==	*(EQ2)*	*[[SWAP - 0=]]*	*... The rarely used one*

Seldom used derivates, usually absent, useful for readability

<>	(NEQ)	[[-]]	... Both signed or unsigned
>=	(EqGreater)	[[<]]	
<=	(EqSmaller)	[[>]]	

Branching Words (internal or routines):

Conditional Branching depend on Equality, opposite to Boolean.

Thus Forth uses uses (0BRANCH) hidden word, for conditional branching, and (BRANCH) for do-it-now.
What about the reverse, N_branch? Reason a bit on the matter, and also on the use of routines vs inline code... and the use of (internal) hidden words (clue: when compiling Forth secondaries).

Note: ACE-Forth does it differently, because disassembly ready. Those must be routines.
However, the ACE can use their word equivalents on ROM words (not disassembled).
Notice structured Forth delivers unconditional and conditional internal Jumps both for JumpForward and JumpBackwards... Find those out!

BTW, (Nbranch) is a simple [[0= (Nbranch)]] compound. Similar on assembler, NZ test is faster.
Any single words are faster. We just point out that (Nbranch) is not essential. (Compromises...)

It makes sense to have both conditional branching routines, when using them. That makes 3 routines.
Why not the double, to cope with forward and backward branching? Relative displacements are an add.

Note: Should not use absolute branches when code MAY be moved (static code is faster, but...)
ACE Forth code is movable, eForth code is static... Compromises!

Atomic Ctrl-Structures

Every Control-Structure is made of a **Test**... And a **Block**, of code to be executed.
The most pervasive: <test> IF <true-blk> THEN <continue>. ((THEN can be replaced by ELSE-THEN))
The simplest cycle: <m-end> <n-start> DO <loop-blk> LOOP <...>

IF-ELSE-THEN, as we may notice, will use Jump-Forward (JpFwd).
Also, that cycles need a Jump-Backward to repeat a block (JpBkd).
    However, cycling may need a front test, as an IF <block> ... terminating with (JpBkd)
    Likewise, another cycling may test at its end, as a <test> IF (JpBkd) <...>

DO-LOOP is simpler in that regard. But it needs a counter, reason why test is made at the end.
Usual practice may also demand a different stepping to be made automatically, ie, at the end.
Note: This ‘stepping’ complicates things. The end-value may be missed (forcing a compromise).

Structure Implementations are quite simple: Addresses are RELATIVE to the Jump, and stored after it.

```
    Jump-Forward  [+x] <...>   will run:   Add +x Cells to IP ... NEXT
    Jump-Backward [-y] <...>   will run:   Add -y cells to IP ... NEXT
```

Tests are still simple: Just react to a value on stack:
    * <u>Boolean</u> ZERO is FALSE... non-ZERO is TRUTH...
But there's a twist: Conventions and ASM collide:
    * <u>Value</u> ZERO is EQUAL... non-zero is MISMATCH...
We must be aware of this, knowing if working values or Booleans.

Short Examples . Structures:

Note-1: JpFwd are unconditional, may be used by a test
Note-2: UNTIL is composed with a test of false for jpBkd
Note-3: REPEAT is an unconditional JpBkd... Test is on WHILE (which is actually an IF)
Note-4: THEN and BEGIN are not needed...   Are marks (for clarity and/or a clear disassembly)

```
                               /----true----\
                              ^              v
    IF-THEN       :  <test>  IF [then-gap]   <IF-blk>   THEN
                                       v               ^
                                        \----false----/


                               /----True----\                     /-------done-------\
                              ^              v......run..........^                    v.....>
    IF-ELSE-THEN  :  <test>  IF [else-gap]   <IF-blk>   ELSE [then-gap]   <ELSE-blk>   THEN
                                       v                                 ^ ...run ......>
                                        \------------false--------------/


    BEGIN-UNTIL         :   BEGIN <block> <test>   UNTIL [begin-gap] ...
                                   ^                              v
                                    \-----<---- ?back? ----<-----/


                                    /-----------<---- !back! ----<-------------\
                                   v                                             ^
    BEGIN-WHILE-REPEAT  :   BEGIN <test> WHILE [Rep-Beg]  <block>   REPEAT [Beg-Rep] ...
                                                      v                              ^
                                                       \------------false----------/
```

Chapter 5 - Structural Elements

Dictionary, on Segments

Forth is an interactive compiler and a self extensible OS. All functions are present on the Dictionary. All compiler data is (also) present on the Dictionary. Functionality depends from Dictionary structure.

<u>An example</u>: If we wish to build Forth independent executables, the Dictionary location must be chosen so to drop Headers. While Bodies are kept available = Need a Code-Dict and an Data-Dict (Headers). Going a bit further, we may need to move words so to join dependencies together = Need mobility. ACE-Forth delivered that mobility (for other benefits). We kept that chance as added benefit.

> *Likewise, we may decide to have 2 Data-Dicts (on universal IND-Forth).*
> *Or 2 Code-Dicts (on SBR-Forth) to keep Kernel Core static and safe.*

The Dictionary is a tool. Less noticed, is that its structure can support more than the Dict itself: The way word searches are done, allow to go from proto-OOP to static-OOP (and further on). Thus, we may define there closed nodes (Vocs)... or define open (Classes).

> How does this affects a Search? It's up to the Search routine to stop or continue on parent level.
> Such is our solution, and its trivial: These root nodes act in the same way, tagged differently.
> Either by their CFA, that could be... either by an header flag (similar to 'Immediate' flag).

As a 'Dict' goes...

Traditionally, the Dictionary is a sequence of Words (Core and extended functions).
Just for convenience, a Word there is described as Header+Body, joined together.
That construction is just a particular 'recipe' only kept while convenient.
Dict versatility, allows to change in size and volume. It's a good tool.

Every 'Dict' is different, all follow a common rule: Even split, Header and Body are linked. Header and Body can reside each on its own partition. Headers are data, can cope with links. Bodies are implemented either as data, either as code. Sometimes as a mix of code and data.

For Code/Data independence, an Header should have an extra Link to locate a Body, in the place of it. This is particularly useful in many cases, whatever the dispatching method or variations (no recipes). When (true) disassembly is used, plus Data/Code splitting is needed, a link-back will ease disassembly.

> For Disassembly, there are several ways to reach a well identified Execution token.
> *ACE-Forth was (still is) unique doing that (as an extra, it's not a usual concern).*

Do 'Words' move?

Traditionally, again, Forth is Disk based. The Dict is on RAM, ready to be replaced.
As such, there's no need to move words around. It's simpler to keep the Dict static.
Without a fast mass storage, or even a slow one, keeping Dict integrity is crucial.

> There are several ways to do that, ACE-Forth choose the most efficient: Relocation.
> This has its own demands, creates new problems (see the Vectors problem on Book #1).
> There's a need for a list of streamed data inside a word (as ." S" and such).

There could be better ways (define 'better') ... Such is the concept of "compromises".

> *The 'better' way... is the one that satisfies our needs. Know your priorities!*

___

 # A friend I travel with years ago, to visit the greatest known 'mamoa' of the Peninsula,
warned half-way: "Pay attention! You are now going to see our new chickens farm"...
(!!!) Wow! Those 'chickens' were tall as a man! Australian they were!

DOES> must adapt

CFAs and DOES> are what makes Forth, not the sequencer. However, the sequencer choice alters things. Each 'mode' or variation has several ways to solve DOES>. CPU Architectures have a big word to say. We do distinguish traditional indirection from the SBR method (open to native code, if we allow it).

Difficulties on DOES>, come from overlooking principles at work... Overlooked under known solutions. As previously mentioned, CREATE-DOES concept is fairly simple (even adaptable):

- Assists generation TYPE definers, referencing User Code. Then reaching it with a Far-Enter.
- User Type definers will build named ITEMS: Will allocate new Dictionary headers, and bodies.
- Item bodies start with a CFA (1 method in that DOES> section, as defined by the programmer).

This TYPE-Definer (with DOES>) plus ITEM-Defined (running the DOES>) is a pair were only DOES> need to respect "Janus-like" Build-time and Run-time. Easily done, adding a DoDOES and a (DOES) routines. DoDoes routine will to place the caller instance address on stack, needed to feed the DOES> section.

Remind the purpose of that method is to be shared: It needs to know were to act, its caller. So far, its simple. The caller starts with its Type Method (CFA). Item location is known. Method location is known (in the CFA). Noe can be lost before the juggling part.

Why tricky? It sounds simple, it is simple. But we have 3 stages to be aware of.
    Step 1 - It's a pair, but the TYPE-Definer must be defined to act as a builder.
    Step 2 - The DOES> run part is defined, to work on the Data Structure Built.
    Step 3 - The Items CFA, running those DOES> actions (common and shared).

We may forget the DOES> word *must* also have a define-time action and a run-time action. This is were people *may* get lost. On build, it must deliver an EXIT to the BUILD part. Then must build the items CFA portion... *(Here things differ, as we will see shortly)*

The apparent simplest thing to do, is to follow the steps and their actions... Not quite:

- Dstk can be the CPU stack, or a Soft stack.
- Code may be mixed with Data, or be separate.
- Threading Method can be Traditional, or SBR/Native.

Any of these will alter HOW to do it, needing a solid WHAT are we doing... After WHY do so.

Again: What is needed... for an Item's CFA to run properly the user-code on DOES> area ?!?
    1 - PLACE the Item address on Dstk, to be used by the Type managing (user) code
    2 - ENTER on that Type code, running it until it exits. Item done!
    3 - The caller of the Item (now done) continues its own sequence

This means (Does) runtime is placed on the definition. While the Item-CFA must do a 'Far-Enter'. Item DATA address is on W, to be placed on stack; class ACTIONS must be executed by the Item CFA. Indirection keeps paying, invoking the DOES> run-time (on the Type definition, as a private CFA). *CPUs allowing "direct mode" will add a CodeField after DOES> calling DoDoes (Item CFA pointing it).*

That's what a Class (or Type) is: <u>A common self-managing CFA</u>, doing a Far-Enter... Brilliant!

Do remind the aspect of DOES> can vary, depending of the implementation (and CPU). Also, but now shorter: A CFA can be a Builder CFA, or a Running CFA. So far, so good. The most standard version is Indirect Forth, made easier when CPU allows to mix code and data. Not all architectures being Von Newman, don't count on those didactic examples. Learn, then forget.

For SBR, the principles are the same. Only the HOW changes.

When DOES> ... What?!

DOES> mounts a user CFA on every (class) Item built.
This way Forth allows the user to expand Forth compiler instead just to compose structures.

It creates new Data types (even compiler structures). To be universal (and by the Forth methodology of leaving addresses of variables and other structures), this is what must be achieved to do so.
- Item invocation must leave the ParamFields Address on stack, identifying Item and Data.
- Type actions must be invoked, address known. Those user defined actions must be executed.

On Item build, is another matter. The Definer is at work, preparing the above Item self-management.
- Creating a new Item, Type definer points its DOES> sections on the new item it creates.
- It also allocates space for the new item. This is the 1st section of the definition.

I.E. DOES> is for the Item built (actions to be invoked by it).
Definer's 1st section, is the builder (an allocator). It just makes sense to have the runner close.

What is at work, how 'does' it work?
We need to recognize 3 steps in the process built: Type Definition, Item Creation... and Item running.
And also, the Proto-OOP self-containment. A Word is independent (unless a Flow Control Word).
Even on common Words we acknowledge a Creation time, and a Running time.
Here we have a Creator creation, a creator running.

To build Data structures (even Control structures) we apparently face another two 'times' by splitting ‘definition’ into 2 levels: Building builders, and using Builders. Thus we need to separate 3 stages:
- (1) Type definition with Build and Run parts...
- (2) Type building... of Items with its new CFA, the above (shared) run part...
- (3) the Item Type (CFA) which invokes that run part AFTER delivering the Item address.

Simple (on goals) but slightly complex, this is were people get stuck (accepting of a sequence shown).
We will say it several times: "A sequence shown is not the goal, it is just a mean." Placing the chariot in front of the horse, is the main reason to get stuck (very didactically).

Not just here. The Theory of Relativity (of time) built by Lorentz and finalized by Poincaré, was also very didactically expressed by E. Maric on her 1905 journalistic works. Used for the abuses of 1919 followed by similar, on the 30's (before greater ones now repeated).

Prior to HOW... (itself prior to WHAT)
We need to ask “WHY !?!” ... The WHAT usually delude us. The HOW usually obfuscate us.
So, we strongly advise: "<u>Know the WHY to build your own HOW, to later get a less important WHAT</u>".

This is were DOES> gets even more interesting:

- Will build new Type variables, beyond (bellow?) compound types (Pascal Records, C structures).
- But also Control words, also to be included in the dictionary (language extensions as CASE-OF).

NOTE: ACE-Forth delivers a variation of Build..Does> ... for 2 different needs and different ‘times’.
It's based on Compile-time (into the Dictionary), versus Run-time (executing its CFA).
(As matter of fact, all control structures are similar to that method. But in ASM.)
We will mention it again, shortly. (Not much, just enough.)

 # Once faking is planted... it makes harder to return, even to become.
We should thank a work fairly done... never clothes suggesting a laird.

What DOES> ... How ?!

How DOES> work, will depend on our sequencer choice.
Also from CPU architecture. We'll try to keep it universal, avoid mixing code and data.

The traditional way (... with Data/Code Split, different from direct mode)

```
Builder  - Running ends placing (DoDoes) as item CFA, followed by Does> part address.
Does>    - Ends Builder Marks a List of Forth xt's to run.
---
(Dodoes) - Push IP to Rstk... New IP := [W++]   (Far-Enter on DOES> location)
           Add 1 cell to W++ (item data location) is pushed to Dstk.
```

All actors are in place for the Item's CFA to run... what the user-defined class does.

The SBR way   (... with Data/Code Split, the hardest choice)

The problem: Item resides on a Data Segment, thus has a CFA, not an SBR CF (Code Field).
How do we solve THAT? ... (There are several solutions... It's an excellent exercise.)

===

And structural words ?!? (... Same thing, different needs)
Instead dealing with data of an independent single body, the purpose is compile inside a word being defined. Thus:

1) Such BUILD..DOES classes <u>need</u> to be IMMEDIATE, to act on compile time.
2) The DOES> action should be compiled <u>into</u> the new word, usually not alone:
3) May need to <u>reserve</u> a small space, to be placed after its invocation (jump data).
4) When the purpose is not a Test->jump, <u>jump over</u> the reserved space (keep coherence).

Common Structural words without testing: [**."**] but also [**S”**] (see Vol#1, “Strings inside Words”)

The above usually demands to know a Forth implementation details.
An exception was ACE-Forth, as it automatised the whole process as a user/programmer benefit.
For that reason it was a bit different. The usual [: name BUILD>] received another name: COMPILER paired DEFINER. While DOES>, also different there, become RUNS> ... (coherence!)

(To be continued)

Chapter 6 - Visible Forth

Last thing before to get a Forth system, is its interactive OS-like nature. (It's a Compiler/OS.)
What is different, what makes it possible, is a simple choice, different from just read and execute.
It's relevance is never noticed. Overlooked because it is mentioned, and simple:

* The user commands interpreter, the micro O.S. interfacing part, allows to use what was compiled, to compile a little more. *Forth (lang) is not an interpreter, it's OS interactive portion is.*

Not just Read-&-Execute
Doing the most with the least, my adopted definition of real engineering, ingenuity (even genius)
... is present on this simple solution that started Forth: The VALUE vs COMMAND dichotomy.

```
- Is it a OPERATION ?   (( Find it... Found? Execute it... Not Found? Expect a Number ))
. Is it a VALUE ?       (( Convert it... Converted?? Push it to stack... Failed? Warn ERROR ))
```

The compiler mode works in similar way... If you are in compiler mode! If you requested it.

```
- Is it a WORD ?        (( Is it a directive, to run IMMEDIATELY, or to compile in the new word? ))
- Is it a NUMBER ?      (( If so and not an error, compile its handler, then the number itself ! ))
```

After that, everything is trivial. This structure allow most Forth core words to be trivial.
Trivial because every Word is easy to build thanks to a private, added 2nd stack.
In other words, what made it so efficient in size and speeds.

> Not hiding neither stacks from the programmer... easing to parsing of function arguments.
> Easing? In relation to Assembler, naturally. But also on the compiler point of view.
> *Managing a Data Stack (a Forth option and detail), is an efficiency option.*

But Forth is not the Sequencer nor the stack.
(It's maybe ... flying over data, directly from Kripton)

Again, what is Forth?
This simplicity, together with the idea of the CFA (self management),
and the resulting added chance to build new Types (self-managed too),
... <u>IS WHAT MAKES FORTH</u> !!!   Then, its OS-like interfacing capabity.

It's an Integrated Set
Building a Forth is only hard when integrating new choices, if present: The visible Forth.
AFTER support is in place, the remaining is recognizable. Everything becomes self-evident.
Or so it seems, when most is already done...

> Because unlimited, CFA based, many possible solutions may be present when using Forth.
> Finding many possible ways to solve a problem with Forth, its advantage and its scare.
> *Its what makes Forth quick to learn, timely to master... Not the stack, but proto-OOP.*
> *We consider Forth the most efficient tool to learn, train and develop... OOP skills!!!*

This integration, IS WHAT GIVES ITS FORM !!! ... *Not to confuse with the CFA mention.*

===

(To be continued)

-

= APPENDIX =

A.1 - The "1%" C. Moore's article

===

1% the Code

This is a provocative statement. It warrants some discussion.

C programs

I've studied many C programs in the course of writing device drivers for colorForth.
Some manufacturers won't make documentation available, instead referring to Linux open source.

I must say that I'm appalled at the code I see. Because all this code suffers the same failings,
I conclude it's not a sporadic problem.

Apparently all these programmers have copied each others style and are content with the result:
that complex applications require millions of lines of code.
And that's not even counting the operating system required.

Sadly, that is not an undesirable result.
Bloated code does not just keep programmers employed, but managers and whole companies, internationally.
Compact code would be an economic disaster. Because of its savings in team size, development time,
storage requirements and maintainance cost.

What's wrong with C programs?

= Some problems are intrinsic to the C language:

- It has elaborate sytnax.
  Rules that are supposed to promote correctness, but merely create opportunity for error.
- It has considerable redundancy. This increases trivial errors that can be detected. And program size.
- It's strongly typed, with a bewildering variety of types to keep straight. More errors.
- As an infix language, it encourages nested parentheses. Sometimes to a ludicrous extent.
  They must be counted and balanced.
- It's never clear how efficiently source will be translated into machine language.
  Constructs are often chosen because the programmer knows they're efficient. Subroutine calls are expensive.
- Because of the elaborate compiler, object libraries must be maintained, distributed and linked.
  The only documentation usually addresses this (apparantly difficult) procedure.

= Others are a matter of style:

- Code is scattered in a vast heirarchy of files.
  You can't find a definition unless you already know where it is.
- Code is indented to indicate nesting. As code is edited and processed, this cue is often lost or incorrect.
- Sometimes a line of code contains only a parenthesis, or semicolon.
  This reduces the density of the code, and the difficulty of reading it.
- There's no documentation. Except for the ubiquitous comments.
  These interrupt the code, further reducing density, but rarely conveying useful insight.
- Names tend to be hyphenated. This makes them unique and displays their position in the heirarchy.
  The significant portion of a name is hard to detect, slow to read.
- Constants, particularly fields within a word, are named. Even if used, the name rarely provides enough information about the function. And requires continual cross-reference to the definition.
- Preoccupation with contingencies. In a sense it's admirable to consider all possibilities.
  But the ones that never occur are never even tested. For example, the only need for software reset is to recover from software problems.
- Conditional compilation. More constants include or exclude code for particular platforms.
  More indentation. More difficulty fathoming which code is relevant.
- Hooks for future enhancements, or abandoned features, are abundant.
  This is useful only in understanding the programmer's ambitions.
- It is in a programmer's best interest to exaggerate the complexity of his program.

= Another difficulty is the mindset that code must be portable across platforms and compatible with earlier versions of hardware/software. This is nice, but the cost is incredible.
Microsoft has based a whole industry on such compatibility.

Forth

colorForth does it differently.
There is no syntax, no redundancy, no typing. There are no errors that can be detected.
Forth uses postfix, there are no parentheses. No indentation. Comments are deferred to the documentation.
No hooks, no compatibility. Words are never hyphenated.
There's no hierarchy. No files. No operating system.

Code is organized so that a block of related words fit on the screen.
Names are short with a full semantic load. The definition of a word is typically 1 line.
Machine code has a one-to-one correspondance with source.

An application is organized into multiple user interactions, with unique display and keypad.
Each is compiled when accessed. Its code is independent, names need not be unique.
A background task is always running.

Comparison

Yes, I could write a better C program that those I've seen. It wouldn't be nearly as good as Forth.
I can't write an assembler program as good as Forth.
No, I don't think Forth is the best possible language. Yet.

But does this add up to 1% the code?
Where is the C program I've recoded? No one has paid me to do that.
One difficulty is comparing my Forth with the original C.
I cheat. The 1% code merely starts an argument that they're not the same.

For example, my VLSI tools take a chip from conception through testing. Perhaps 500 lines of source code.
Cadence, Mentor Graphics do the same, more or less. With how much source/object code?
They use schematic capture, I don't. I compute transistor temperature, they don't.

But I'm game. Give me a problem with 1,000,000 lines of C. But don't expect me to read the C, I couldn't.
And don't think I'll have to write 10,000 lines of Forth.
Just give me the specs of the problem, and documentation of the interface.

My Conclusion

colorForth's incredibly small applications provide new estimates of their overstated complexity.

===
<End of Text>

____
This Article is kept on https://colorforth.github.io/

A.2 - Forgotten details

Disassembly (and the Dictionary)

Forth is a "Compiler Online', always ready.
The dictionary stores that FORTH compiler data. It is one of Forth bases for flexibility.
It allows code expansion and management. Not disk supported, ACE-Forth allows word replacements without the need to trim and reload what follows a changed Word. Words will ‘slide’ on the Dictionary.

There are many alternative implementations to allow disassembly... There can be no equivocations.
The ACE dict structure is fairly common, only with a slightly difference position of elements (for smaller code). Yet, as ACE-Forth adds true disassembly, extra fields are needed to deal with COMPILER build words and disallowing equivocations. So the dictionary can also slide.

A few words on Decompiling a Word and sliding the Dictionary (invoked by the pair LIST/EDIT)
In principle, it 'would' be easy to decompile when code is threaded... if only Forth words are used.
Regardless of a dispatcher, Address 'threaded' would identify a word, allow both Disassembly and Slides.

Forth introduced pre-OOP characteristics, code sharing (types) with DOES> mechanism.
This introduced an indirect decompilation. And another, with the RUNS> mechanism.
When Forth has his own dispatcher, instead subroutine threaded, OOP is natural.

More than that, 'redefining' is architecturally related with 'decompiling' (same basis).
There too, there are several way ways to do it. Again, Dictionary structures are affected, the ACE solution only demanding the decompiling structures mentioned above. But, there's one, at the expense needing a list of exceptions for COMPILER words. We may guess this would be copied to user space, to keep the solution fast and clean.

A Dictionary structure depends from a particular implementation goals.
Bigger changes are needed when developing different ‘modes’ (subroutine and/or native code)
These show particular needs. As when toggling between interpreting and compiling. Or needing to deal with Code and Data separate areas (CPU arch demanding, or due segmentation).
Here's a short image of other variations (not including SUBR nor NATIVE code):

```
    Joined implementation, each word seen as a block of 2 sections (on Code=Data)
        + Dict Header   [ name + ^previous ]
        + Word Body     [ CFA (or CF) + Data managed by that CFA ]
    Words are in sequence as in   Whead+Wbody + Whead+Wbody + Whead+Wbody

    Segmented implementation, Header -> (Primary or Secondary Word) ~~~ An example
        Data Segmt :    Headers with [name + ^previous + ^Type + links]
        Code Segmt :    [ Primary Words CPU code ]
        Data Segmt :    [ Secondary Forth code ]

    Other, as Native Compiling Dictionaries are slightly different from Dispatcher compiling.
        The idea is the same, just adapted to the situation. Many things are harder. Thus:
        What drive implementations is... the engineering goals. Not technical 'recipes'.
```

Remind 'copy' of existing 'solutions, adds very little. 'Groking' is needed (seen on the ACE).

___
\# "The highest form of human intelligence is to observe yourself without judgement." ~ Krishnamurti

A.3 - Grains of Salt

All we learn is incomplete. *Seldom plain wrong, built to fill absences. Or as 'Mantras' to invoke.*
Many theories and deductions are built as 'Egyptology'. *#Or worse, as a gossip driven 'medicine'.*
NOR is there any Science on linear arguments to be repeated. BUT on Sight, Honesty and Courage.
 # Paradoxically (opportunistically) Sight and Courage ARE ABSENT... on most present references.

Training is Mimicking. Discovery demands much above. Experimentation is just an aspect of Honesty.
As a personal path, 'ConScience' needs Courage, so to face 'education' (known as a fast-food).
 # In spite of the damage installed, many can still laugh (Humour helps, more than Sarcasm).

1 - Physics *(A few things we find useless to explain, as 'interests' rule over reason)*

a) Present concept of Time is as wrong. As the obsession with gravity.(It's a deaf subject)
b) The Hypothesis of Black Holes building up, overlooks Time: Gravity vanishes too!...Oops!
c) The Hypothesis of 'Big-Bang' is a Work Hypothesis. Imposed as a Theory, it's now 'theology'.
d) The derivative equations after Poincaré, made by Einstein-Maric (then attributed to her husband) are incomplete due misunderstanding. Two terms missing: The 1st, allowed us to predict the Voyager 'slide' (2 years before known). The 2nd is an environmental variable, now distorted on Black-Matter delusion.

2 - Philosophy, Psychology and 'Education' *(Beyond dogmas and their acceptance')*

Fact: You cannot pull a black cat from a room in the dark, when that room is only supposed:
a) Mankind is not rational. To mimic such above-nature is a dangerous disrespect for itself.
b) So called IQ measures neither Intelligence nor Conscience. It measure different smartness's.
#My dog was a true Scientist. Most my College teachers were not. He questioned, they played.
c) Artificial Intelligence, being deterministic, is not such. *#Do make impressive thermostats.*
d) 'Education' is mostly training. Information is not Knowledge, as Knowledge is not Wisdom.
#Filling a Turkey, replacing 'Spirits' with information, gives another drunk turkey kind.
#These, sometimes, showing a Prima-Donna symdrom. (Sometimes, an Adolf-EinStone symdrom.)
e) In general, History evidences "accepted knowledge" flaws. Much late, sometimes too late.
'Accepted', makes 'our' virtual reality 'culture' blindness. Of "Believe, it's written".

3 - History *(A few images, from hundreds more)*

a) Civilisation did not started 6000 years ago (mainly chosen to satisfy "Creation Day").
b) Giza Pyramids are not Egyptian. Old Egyptians (ignoring new Egyptology) have told us so.
c) The head of the Sphinx is obviously made of concrete, rebuilding an older 'feature'.
d) Easter Island monoliths are faces with bigger heads lost, as result of a softer rock:
Heads were red, similar to remains found in Paracas(Peru). Certainly NOT Russian hats.
e) The end of the Euro-American Ice-age corresponds to the start of PRESENT Siberian Ice Age.
Mammoths stomach contents say so!. To 'experts', continents are fixed: Earth is rectangular.
f) Plato Atlantis references are to a 'continent'. PLUS a didactic event, much recent, we known as the Minoic demise (reference was as the city in a round crater). Obviously, NOT the same.

4 - Medicine (Because more 'popular', it's now *more like a church than Astro-Physics.*)

a) Cancer theory is now forcing a genetic supposed connection. (just politics, not Science)
b) Virus theory still is an hypothesis. When tested, it failed. (such failures were omitted)
c) On Lab analysis, anonymous 'blind' samples are crucial for fair results. Now dropped. Why?
d) Similarly, Medical 'files' become a source of stigma. Sometimes used to discredit, spread distorted 'interpretations'. Id based.gossip replaced diagnosis and needed fair treatment.
e) *#"Brave New World" arguments try to 'dispose-off' "the other" 6 Billion, seen as 'rivals'.*

5 - The 'Legality' Onion *(Forgotten under the spell of "we are accustomed")*

a) Legality impetus was once driven by Ethics. Now distorted by Propaganda (Psychology based).
b) Natural 'law' become surrounded by successive layers of increasingly arbitrary constructions.
c) 'law' and democracy', when supposed, are ladders to submission (T.Jefferson knew 'dynamics').
d) Not even Nature's 'laws' are Laws, but our interpretations only. ~Richard Feynman (by sense).

 # Laugh as any Butterfly laughs, across Oceans, across Time: Not 'learned', they do fly.
 # "Whom gets close to children, sooner or later will get pissed" ~ Portuguese proverb
 # We should never share what will not be understood. Burning poles say so. Face it!

• • •

--- End of Book 3 ---