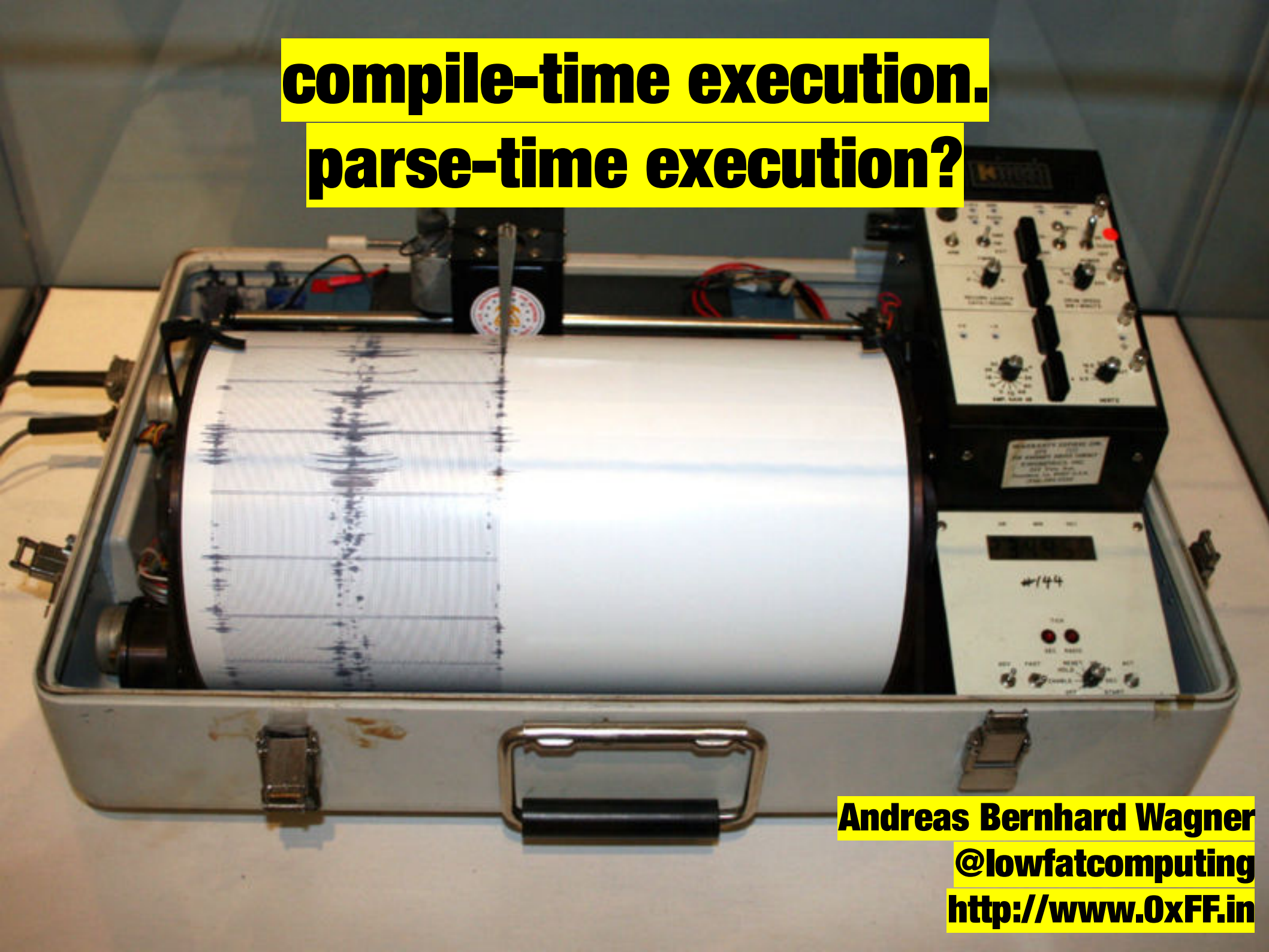

compile-time execution.
parse-time execution?
Andreas Bernhard Wagner
@lowfatcomputing
http://www.0xFF.in

Forth for general purpose computing?

unplanned computing

~~Forth for general purpose computing?~~

Forth vs. Unix

- **Unix** commands are **filters**
  - Function **composition.** Separate data from code.
  - Plays well with others. **Ecosystem oriented.**
  - One-off **disposable interactions,** but **complex code.**
  - Intended to be **general purpose.**
- **Forth** words are **abbreviations**
  - Function **juxtaposition.** Data is more often inlined.
  - One-off **disposable code. Interactions** take **more effort.**
  - **Self-sufficient** systems, that is.
  - For **single-purpose** embedded systems.

Forth for unplanned computing:

- random access over problem/solution
- make Forth better at filtering

When someone asks how to parse a csv file in Forth...

...I say: use the comma compiler

```
CREATE csv ( age , height)
                4 , 41 ,
                5 , 44 ,
                6 , 47 ,
                7 , 49 ,
                8 , 50 ,
                9 , 51 ,
```

"*in Lisp code is data; in Forth data is code.*" - Sam Falvo

all input/output as runnable forth code

- Welcome string resets
- No OK prompt

  (use nl character)
- If **unrecognized**: comma **compile it**.
- If **partially unrecognized**: **define something** with it.

```
gelForth ( v0.1)
2 3 + . 5
```

*if **data is code**, we'd be **filtering code**.*

When someone asks how to parse a csv file in Forth...

...I say: use the comma compiler

...they say: my csv file looks like this:

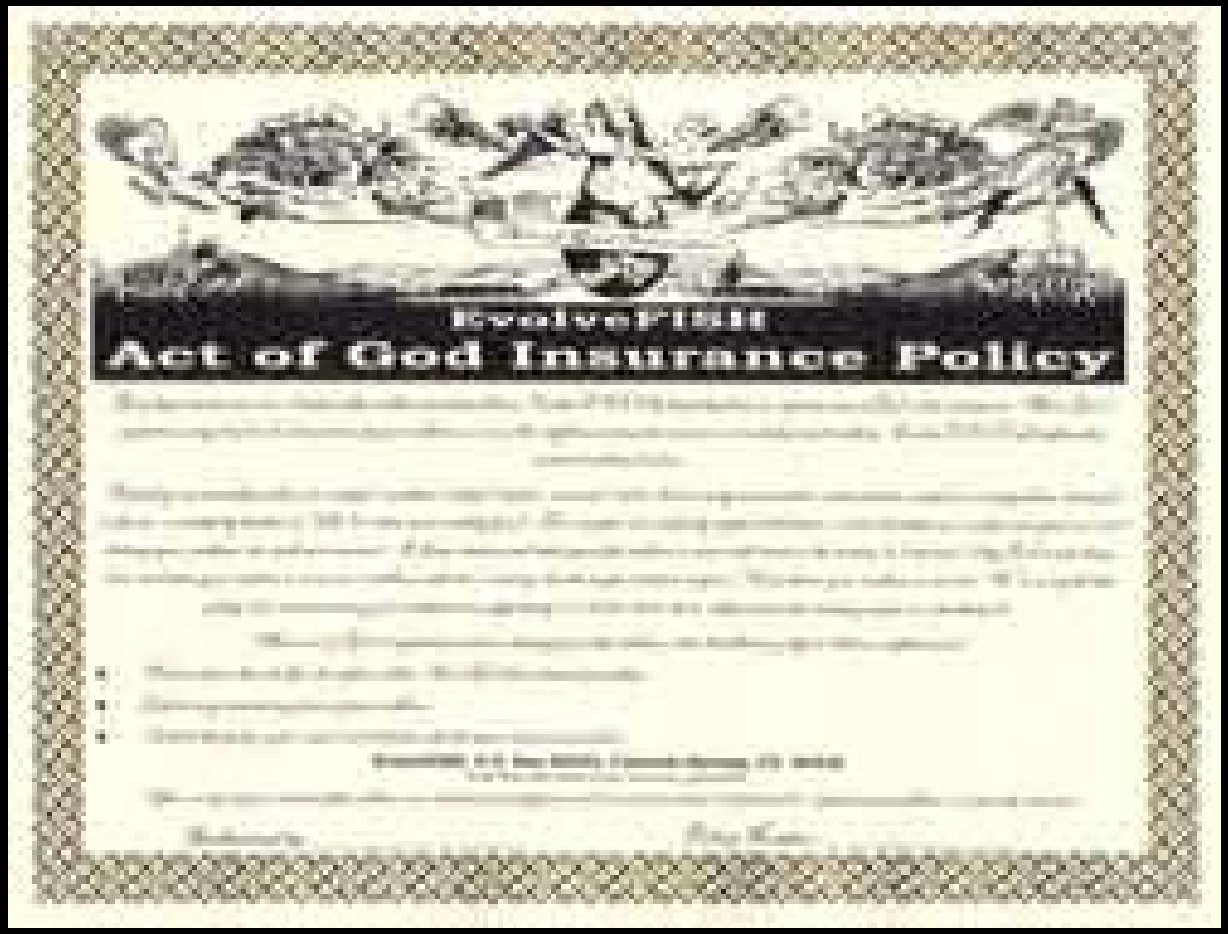
...I say: yeah...data only comes in words.

```
age,height
4,41
5,44
6,47
7,49
8,50
9,51
```

Sometimes what I need to parse is beyond my control...

- non-Forthers designed a format.
  - (this happens a lot)
- The external natural world (beyond the uC sensors).

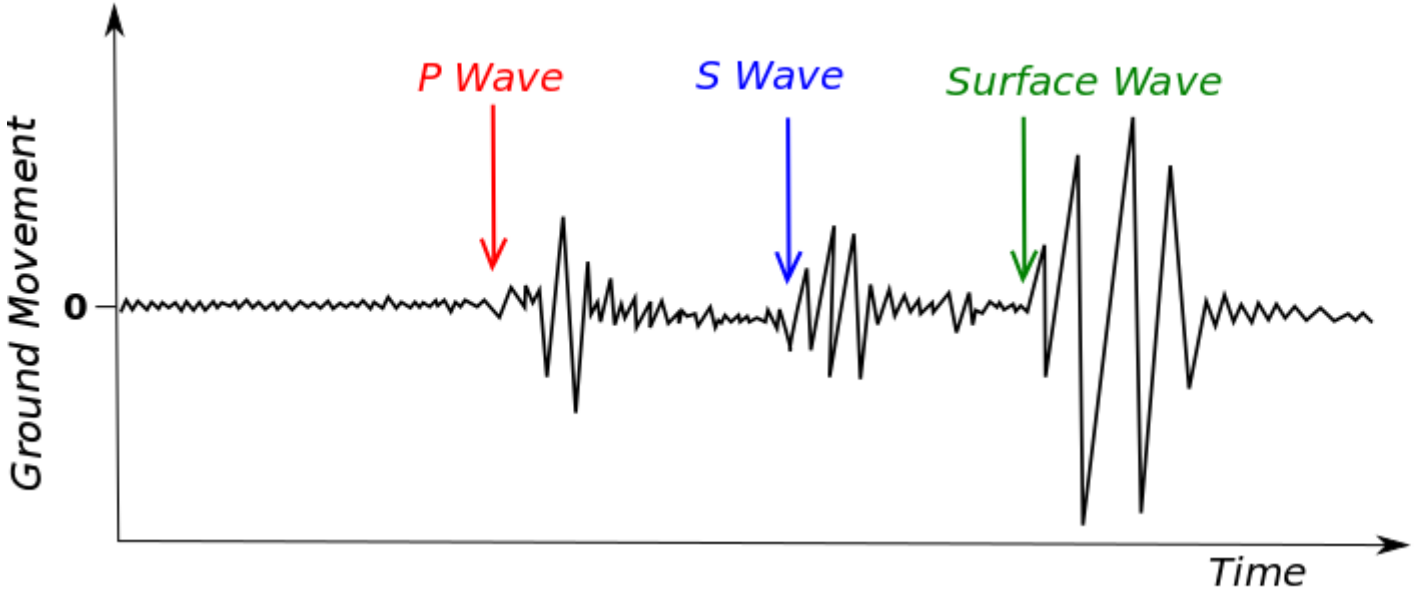
Your insurance calls it: "The Act of God Policy"



Parsing:
it's not their fault, but it is my problem

- Comprehension of seismograph output as part of your forth's lexicon.
  - **Earthquake readings are not literals.**
- Non-texual sensor readings are **data**.

Shouldn't they also be code?



Seismometer

[–] **thamesynne** [+4] **6 points** 3 months ago* (last edited 3 months ago)

Consider

```
: >DIGIT 48 - DUP 9 > IF 7 - THEN ;
```

Don't worry about the magic numbers; the important thing is the `IF .. THEN`. Without immediate words, `IF` and `THEN` would just be compiled as normal words... and `IF` would have to scan ahead for `THEN` and resume interpretation from that point. Sounds doable... but then consider:

```
... IF ... IF ... ELSE ... THEN ... IF ... THEN ... THEN ...
```

Now `IF` has to search ahead not only for any old `THEN`, but for the `THEN` that matches it - or for an `ELSE` that matches it - which means keeping some form of bookkeeping as to which one it's doing. And that bookkeeping has to be re-entrant. That's a lot to do at runtime, especially in bits of code where performance is likely to be a critical consideration!

However, with immediate words, you can define IF as a word that lays down a conditional branch with a placeholder destination, and THEN as a word that fills in that destination with its own address. It's a lot easier.

Of course, it's not the only possible approach. If Forth had block structures - so for example, you could write

```
{ 10 0 DO I . LOOP } EXECUTE
```

and have the digits appear on your terminal - you could define IF, and other control structures, as combinators, so the above word would become

```
: DIGIT 48 - DUP 9 > { 7 - } IF? ;
```

But that brings up a chicken and egg problem: How do you define `{` and `}` without immediate words? *With* immediate words, it's easy: `{` stores the address of the code being compiled, `}` retrieves that address and shuttles that section of code off to the top of memory, leaving its address on the stack or compiling it inline. *Without* immediate words, you're back to scanning the code stream at runtime for `{` and `}` and counting nesting levels.

And then there's the word that's in every definition, but whose function is always forgotten: `;`. If you don't have immediate words, this needs to be specifically searched for by the compiler, in order to stop compilation and return to execution mode. And the compiler needs to worry about what happens if it runs out of input before seeing a `;` - although at least nesting isn't a thing in standard Forth. However, there's also `[`, which puts the compiler on hold for a while - so now the compiler is searching for two special characters - and that set can't be extended without rewriting the compiler. (This is how Lisp REPLs were written before reader macros, incidentally. Forth's immediate words replace reader macros, rather than fexprs or DEFMACROs... and neatly demonstrate that reader macros, rather than Lisp macros, are necessary and sufficient for an extensible language.)

So avoiding immediate words leads to an awful lot of special-casing all the way through the entire interpreter structure; it also means that that interpreter structure ends up unchangeable, and almost certainly can't be defined from within Forth itself. And it takes a *lot* more code. Even adding one immediate word eases the load considerably... and once you've added one, why not make a mechanism that lets you add as many as you like? And suddenly, everything gets much easier.

And just to screw with your mind a bit more: consider that IMMEDIATE is generally *not* an immediate word - and how that could be put to use...

Could the idea behind IMMEDIATE words also dissolve parser complexity?

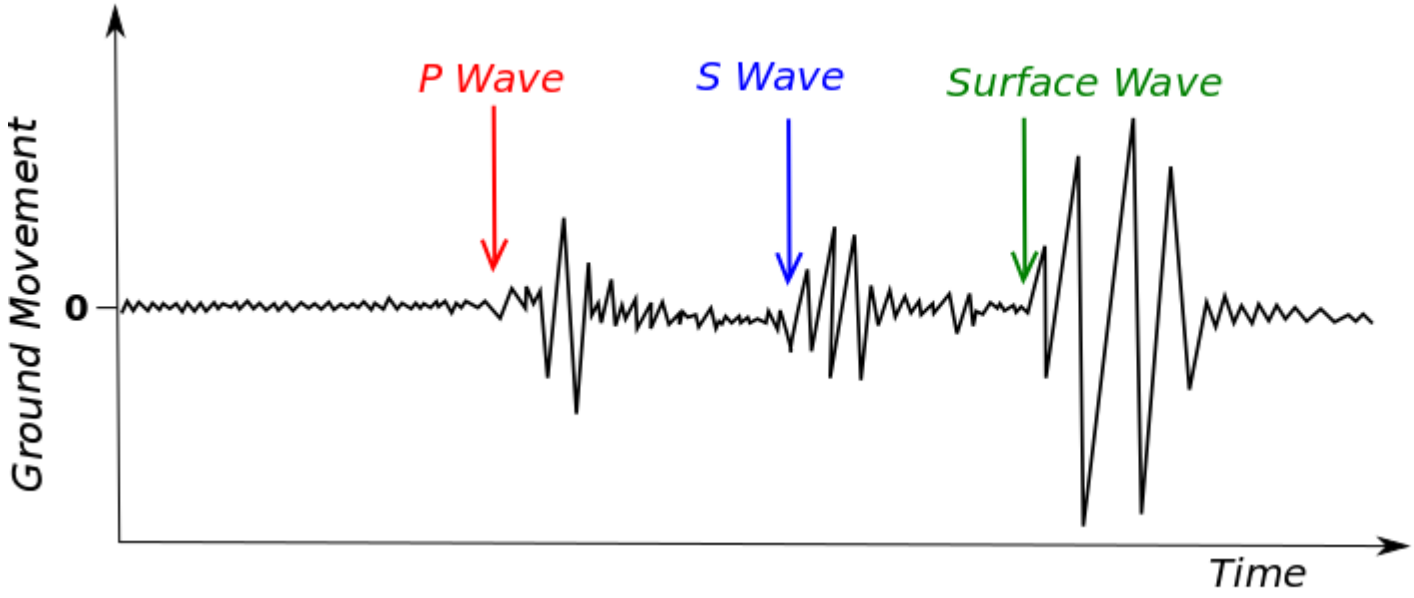
The Forth code is the user interface.

My keyboard is a user interface...

...But so is my mouse

Could the mouse write code?

A word named



That executes only when we have an earthquake?

pattern matching as the inverse of functions

1-	1+
swap	swap
2/	2*

(applications in reversible computing)

http://micsymposium.org/mics_2009_proceedings/mics2009_submission_72.pdf

Put code in place of the ascii string header

one ! and one @

~~**C! C@ !!! H@**~~

@ (match “!”)

! (match “@”)

$FF <= (match number)

@ (match “@”)

! (match “!”)

$FFFFFFFFF <= (match number)

examples:

‘ still finds xt.

headers are typically written to be:

non-destructive.

...maybe also the body?

$FF ‘ (xt for bytes)

$FFFF ‘ (xt for hcells)

2 4 ‘ (xt for double)

pattern matching with the dictionary: mind the (hyperstatic) scoping

- **@** a word and **!** it into another to get its scope.
- /u/dlyund’s (Mark Smith’s) idea.

BIND + parsing-state execution?

$FF ‘ @ $FFFFFFFF ‘ !

(replace 32-bit semantics with 8-bit semantics by manipulating scoping)

Thank you!

Andreas Bernhard Wagner
@lowfatcomputing

/u/dlyund's BIND word

[–] dlyund +59 [F] **3 points** 1 year ago* (last edited 1 year ago)

> I'm not sure how to implement bind in gForth and I want to avoid implementing a new Forth for Linux in order to write another one for the PSoC. So I'd be interested in any thoughts on how to do this or other questions, comments, and world views.

I recently implemented `bind` for a more traditional Forth by adding a field that points to body of a definition to the header.

```
+------+
| link |
+------+
| body |
+------+
|      |
```

`bind` can then simply fetch the body from one definition and store in another.

```
: bind word body @ word body ! ;
```

It's used as follows

```
bind text.utf8.same same
```

This approach had second, minor, benefit. Since the body field is placed directly after the link field the address of the body doesn't have to be computed; when creating the header the body is set to the value of here before exit, and the body of the definition get's compiled after.

Did you try to implement this yourself? If so I'd be interested in other solutions.

How is gelForth progressing?

permalink source embed unsave spam remove give gold hide child comments

references

- http://amforth.sourceforge.net/pr/Recognizer-en.pdf
- http://micsymposium.org/mics_2009_proceedings/mics2009_submission_72.pdf
- http://forth.wodni.at/wrongforth
- https://hub.darcs.net/pointfree/forth-bind