Forth als Basis für einen portablen Assembler

M. Anton Ertl
TU Wien

Portabler Assembler? Was und wozu?

- Traditionell: Assembler als
  Zielsprache für Compiler und
  Implementierungssprache für Interpreter

- Nachteil: Assembler ist nicht portabel
  Compiler muss für jede Maschine geändert werden
  Interpreter muss für jede Maschine neu geschrieben werden

- $\Rightarrow$ Viele Native Code Compiler können nur ein Target, oder wenige
  Die meisten Interpreter nicht mehr in Assembler

Warum nicht C?

- C erfüllte die Aufgaben eines portablen Assemblers

- Viele Interpreter wurden in C geschrieben (z.B. Gforth)
  Einige Compiler verwendeten C als Zielsprache (z.B. GHC)

- Aber: Im C-Standard gibt es viele Löcher
  z.B. ganzzahliger Überlauf

- C-Compiler benutzen immer mehr Löcher zur Optimierung
  Sie optimieren Programme kaputt
  Kein direktes Verhältnis mehr zwischen Sprache und Maschine

Was tun?

- Ein vernünftiger C-Compiler?
- Warum an einem Compiler für C arbeiten,
  wenn ich Forth implementieren will?
- Forth ist low-level, gute Basis für portablen Assembler
- Manche Features von Forth passen nicht ganz
- Ein Subset von Forth mit einigen Erweiterungen
- Python→RPython, JavaScript→asm.js ...
- Aber bei Forth passt die Basis besser

Gewünschte Eigenschaften

- Portabel: Funktioniert auf verschiedenen Maschinen
- Ausdrückbar, was die Maschinen können
- Direktes Verhältnis von Sprachfeature zu Maschinenfeature

Zielmaschinen

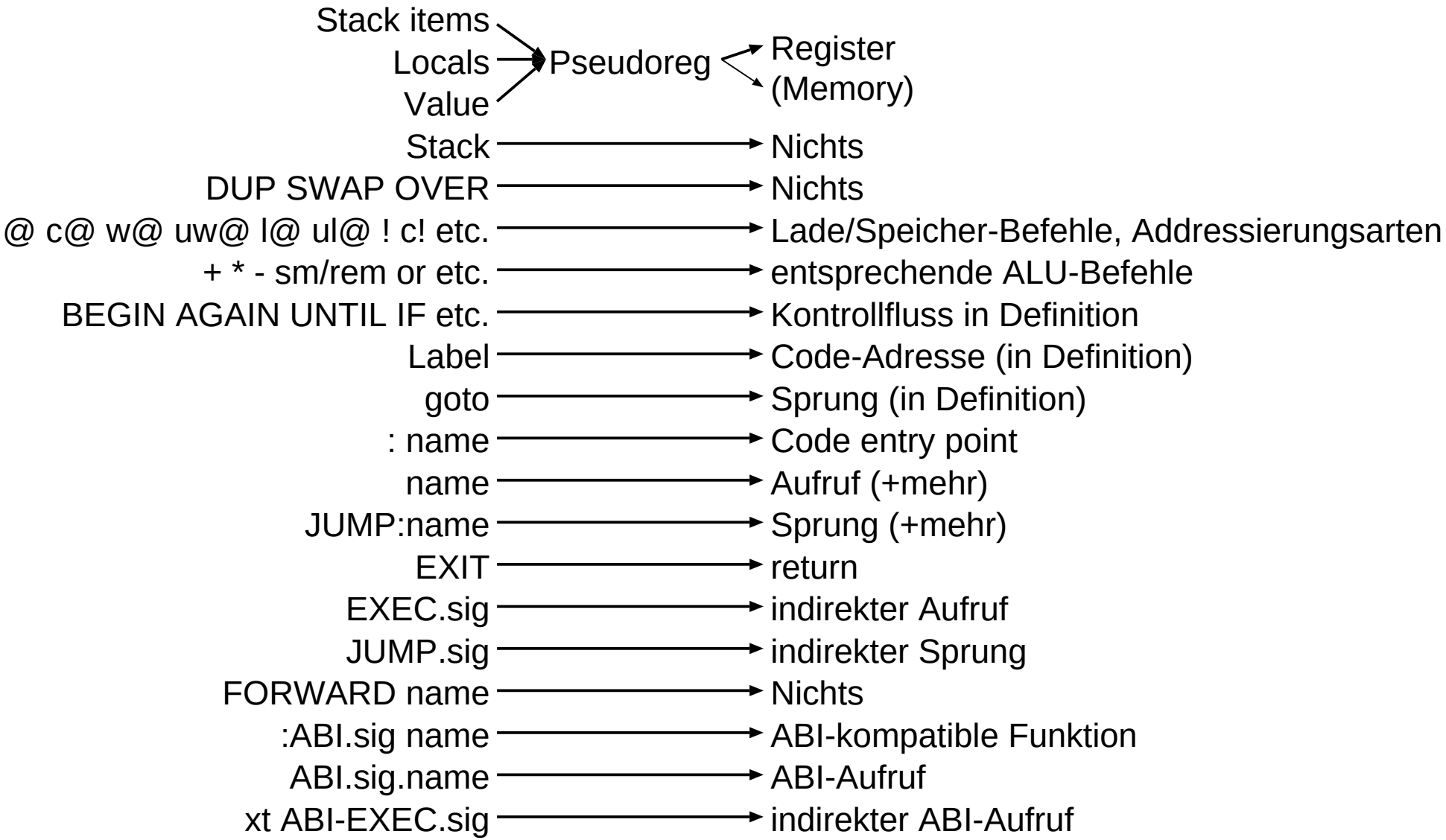
- General-purpose computer architectures
- Byte-addressiert mit flachem Speicher
- 2er-Komplement-Arithmetik auf Wortbreite

Beispiel

```
: max
  2dup > if
    drop exit then
  nip exit ;

:abi.xx- printmax {: n1 n2 -- :}
  "max(%ld,%ld)=%ld\n\0" drop
  n1 n2 2dup max abi.xxxx-.printf
  exit ;

\ Aufruf von C:
\ main() { printmax(3,5); return 0; }
```

```
 cmpl %edx,%eax
 jle L28
 ret
L28:
 movl %edx,%eax
 ret
```

PA-Forth vs. Maschine

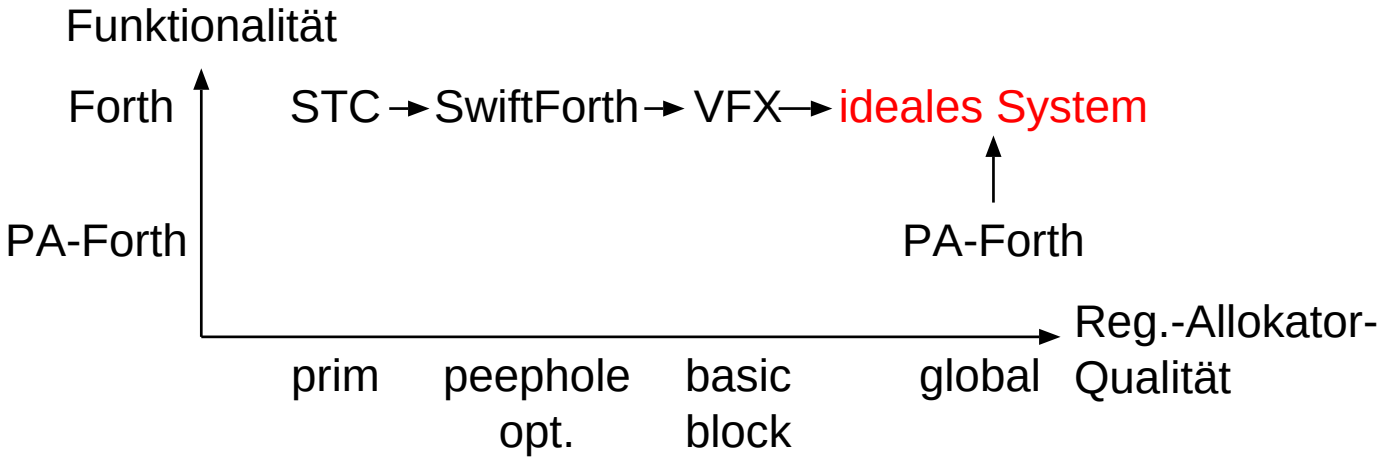


PA-Forth vs. Forth

- Stack items immer statisch bekannt
  Keine unbalancierten Stacks
  Eingeschränktes EXECUTE

- Höhere Forth-Features nicht nutzbar

Von PA-Forth zu Forth



- Würde Chuck Moore es tun?
- Forth→PA-Forth Compiler?
  Stack-Analyse doppelt
  ... aber illustrativ
- PA-Forth zu Forth erweitern
- oder PA-Stackanalyse für Forth-Compiler zugänglich

Von Forth zu PA-Forth

```
\ Forth

: do-method ( .. obj m-off -- .. )
  over @ + @ execute ;

: foo ( .. obj -- .. )
  1 cells do-method ;

: bar ( -- )
  1 2 my-obj foo . ;
```

```
\ PA-Forth
0 value sp

: do-method
  over sp cell- tuck ! to sp
  swap @ + @ jump.- ;

: foo
  1 cells jump:do-method ;

: bar
  sp cell- 1 over !
     cell- 2 over !
  to sp
  my-obj foo
  sp dup @ swap cell+ to sp
  jump:. ;
```

Von Forth mit Return-Adressen zu PA-Forth

```
\ Forth mit RA-Manipulation

: do-method ( .. obj m-off -- .. )
  over @ + @ execute ;
```

```
\ PA-Forth
0 value sp
0 value rp

: thunk1
  exit ;

: do-method
  over sp cell- tuck ! to sp
  swap @ + @
  rp cell- to rp ['] thunk1 rp !
  exec.- rp cell+ to rp
  jump:thunk1 ;
```

```
: foo ( .. obj -- .. )
  1 cells do-method ;
```

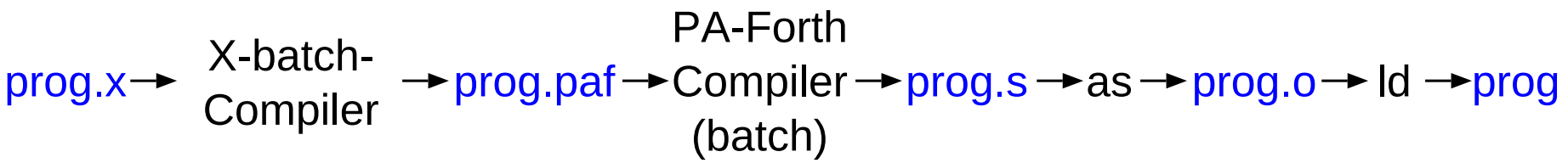
```
: bar ( -- )
  1 2 my-obj foo . ;
```

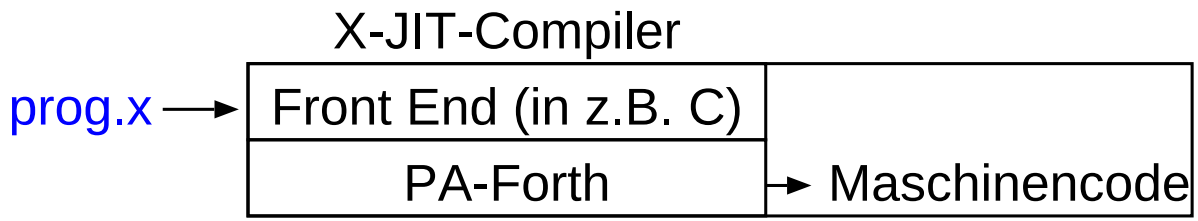
```
: foo
  1 cells
  rp cell- to rp [’] thunk1 rp !
  exec:do-method rp cell+ to rp
  jump:thunk1 ;

: thunk2
  sp dup @ swap cell+ to sp
  jump:. ;

: bar
  sp cell- 1 over !
     cell- 2 over !
  to sp
  my-obj
  rp cell- to rp [’] thunk2 rp !
  foo rp cell+ to rp
  jump:thunk2 ;
```

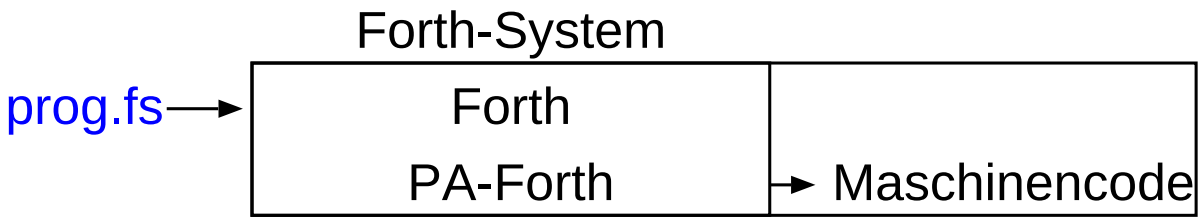
Offene Fragen

- Garbage Collection etc.
- Standard-Debugger
- ... aber C war erfolgreich ohne diese Features
- Exceptions
- SIMD-Befehle

Offene Fragen: Wie compilieren?







Verwandte Arbeiten

- `C--`
- GNU Lightning
- LLVM
- IL (für APL)
- Sprach-Subsets:
  RPython (PyPy)
  asm.js (JavaScript)
  Pre-Scheme (Scheme48)
- Machine Forth

Zusammenfassung

- Forth-Variante als portabler Assembler
- Direktes Verhältnis Sprache:Maschine
- Als Forth: Benötigt nur einen stack pointer
- Zeigt teure Forth-Features auf