LISP Hardware

Was steht hier eigentlich in den Räumen rum?

Was ist LISP?

- Eine der ersten Programmiersprachen
  - 1958 von McCarthy am MIT erfunden
  - Sehr viele Varianten
    - LISP 1.5, MacLISP, ZetaLISP, Scheme, Common LISP, Franz LISP, etc…
- Basiert auf dem Lambda Kalkül
- Sowohl funktional als auch imperativ
- Später auch noch OO

Warum ist LISP cool?

- Alles ist dynamisch:
  - Code schreiben, in das laufende Programm einfügen, testen, anschauen
- LISP verarbeitet Listen
  - Aber! LISP-Code ist selber eine Liste
  - LISP-Code kann LISP-Code generieren
- Überleetes Macrosystem

LISP ist doch gestorben!

- Emacs LISP
- AutoCAD
- Yahoo Store
- Prozessordesign
- Web-Applikationen

Wie sieht LISP aus?

- Alles ist eine Liste (oder fast, auch Arrays, Hashtabellen, usw…)
- Funktionen definieren:

```
(defun blorg (x) (mach-was-mit x))
(defun blorg2 (&key (x 5) (y ‘symbol) &rest
  body) (mach-was-mit x y body))
```

Wie sieht LISP aus?

- Variablen definieren:

```
(defvar *variable*)
(setf *variable* 5)
```

- Aber auch lokale Variablen:

```
(let ((x 5) (y 6)) (mach-was-mit x y))
(let* ((x 5) (y (+ 5 x))) (mach-was-mit x y))
```

Wie sieht LISP aus?

- Listen sind wichtig (erstaunlicherweise!)
  - Liste besteht aus Paaren
  - Das zweite Element des letzten Paares ist die leere Liste (nil):

  (cons 1 (cons 2 nil))

  Oder auch: (list 1 2)
  - Mit cdr und car auf Paare zugreifen

Was ist an Listen cool?

- Man kann über Listen prima iterieren:

  (dolist (l ‘(1 2 3 4)) (princ l)) = 12345

  (mapcar #’princ ‘(1 2 3 4 5)) = 12345

  (loop for l in ‘(1 2 3 4 5) collecting (+ l 1)) = (2 3 4 5 6)

  Usw…

- Man kann Listen als ADTs benutzen:
  - Stacks, queues, etc…

Warum ist Emacs cool?

- Programm läuft unten, Code oben
- Evaluieren von Code in das laufende Programm
- Aufrufen von Dokumentation
- Completion, usw…

Symbolics

- Geschichte von Symbolics aus wikipedia holen

Symbolics 3600

- Zetalisp
  - Flavors (Objektorientierung)
- Coole Hardware (Grafik, Maus, Netzwerk, Sound)
  - Windowsystem
- Unterstützung von LISP in Hardware
  - Tagged Data
  - Hardware Garbage Collection

Programmierumgebung

- Editor (Zmacs)
  - Indentierung (!)
  - Dokumentation, Completion
  - Kompilierung von geändertem Code
- Listener
- Debugger
- Inspector, Class Browser

Zetalisp

- Common LISP noch nicht standardisiert
- Loop-Konstrukt
- Objekt-orientierung
- Macros
- Package-System
- Aber auch andere Sprachen:
  - Interlisp, Fortan, LIL

Macsyma System

- Anfang der 1960er, Mathematik Software
- 1 Gbyte VRAM
- Hochauflösende Grafik

Symbolics 3600 Hardware

- 32/36 bits Prozessor
  - 4 bit Tagging
- MC68000 als Front-End Prozessor
- Instruction Prefetch
- Microtasking
- 20 Mbytes/s I/O Leistung
- 1 Gbyte virtueller Speicher

Hardware Typchecking

- Tagfield in jedem Datenwort
  - Beispiel: int:5, real:4,2
  - Add int:5 real:4,1 = real:9,1
- Generische Assemblerbefehle
- Effiziente Kodierung von Listen
  - CDR Packing

Kellerspeicherarchitektur

- Kellerspeicher benutz für:
  - Funktionsaufruf (control stack)
  - Variablenbindings (binding stack)
  - Datenspeicher (data stack)
- Data stack wird benutzt, um Garbage Collection zu vermeiden

3600 Instruction Set

- Kein Assembler, dafür zeigt der Debugger LispM Assembler
- 17 bits Instruktionen (9 Opcode, 8 Operande)
- Mischung aus Hardware und Microcode
  - Werden 2 ints addiert: Hardware
  - 2 bignums: microcode

Microtasking

- Gleichzeitig I/O und Rechnen:
  - Microtasks für
    - Plattenzugriff
    - Zetalisp Emulator
    - Netzwerk
    - Sound
    - Front-End Kommunikation
    - DMA

Memory Management

- 44-Bit Wörter:
  - 36 Bit Daten, 7 Bit ECC, 1 Bit bl0rg
- Virtueller Arbeitsspeicher
- Cache Architektur

Garbage Collection

- Pointertyp in Hardware checken
- Markieren von Speicherpages
  - Markiert temporären Speicher
  - Einswappen von Seiten die temporären Speicher haben