The MakerLisp Machine:
A New Platform For Classic Software
7/16/2019

makerlisp.com

Outline

I. MakerLisp Machine

A. What is it ?

B. Why ?

C. System parallels with early 80's microcomputers

D. Modern materials and methods used

II. MakerLisp

A. Why Lisp ?

B. What is MakerLisp ?

C. How the language choice influenced the system design

D. How the system design influenced language implementation

E. Fundamental computer science education

III. Why is this a good machine for other things too ?

A. CP/M, CP/M-ish, DOS-ish

B. Nuttx, or other Unix-ish OS

C. Straight monolithic embedded work

IV. Q&A, some demos

What is it ?

A very simple personal computer

Modern construction, contemporary interfaces

Familiar “vintage” feel

~1985 processor /OS functionality / performance

Inexpensive, modular, portable

Maker-friendly

Why ?

- I miss the old PCs, DOS, CP/M, etc.
- I like the Maker & Retro movements, I like Lisp
- Make the machine I want for the system I want
- Make the system I want for the machine I want
- With the right recipe, we DON'T need more than 640 K (but we can have 16 M if we want)
- Tired of sacrificing quality for the sake of standards / compatibility, I think we can do better
- Freedom from the cost of conformity

Like early (micro) computers ...

- Text, single thread UI metaphor
- Simple, real, memory model
- CPU, terminal, disk, I/O
- Very narrow connection to “OS” and machine
- Trivially portable
- Your program does, and owns, it all
- Or it's just nice simple hardware for any OS

… in a modern computing fabric

- 50 MHz micro-controller/processor SOC
- 4 layer SMT business card
- USB/UART console, USB keyboard
- SPI uSD
- uC GPIO
- High density Hirose expansion board connector
- Jumper-less configuration sensing, control
- Low power (150/250 mA CPU/full system)

Why Lisp ? What's Cool ?

- Everything just fits, “working code” works everywhere, every time. Done, move on.
- Understand what your program is doing, in the abstract and concrete, reliably reason about it (but sometimes it's like doing integrals)
- Brevity, high semantic “energy density”
- Closures, continuations, and macros
- Smallest/easiest functionally complete system

MakerLisp Details

- Small, light, fast
- SECD, Scheme evaluation model
- Written in C, and Lisp (functions, macros)
- Blend of Common Lisp, Scheme, and C
- Very good for very small machines
- Target Vintage Embedded Makers
- Give Forth and CP/M fans something fun to play with. Runs on Linux, too

Machine

- 50 MHz eZ80: uP with uC-style peripherals/GPIO
- Business card / expansion board
- CPU / terminal system
- VGA with 64 color code page 437 text display
- USB keyboard
- Good for CP/M or embedded cross dev, too
- Not Arduino, not Raspberry Pi, not IOT
- Modular, breadboard-connected, 1980's PC

Language

- MakerLisp Quick Reference
- No strings, just symbols
- cats, car, cdr on symbols
- (eval expr k)

“Low level” Macros

- Can boggle the mind, but
- Universal program/language extension tool
- As long as you stay in Lisp's (nearly no) syntax
- A macro is a Lisp function that creates a Lisp expression, from its (unevaluated) arguments
- And then evaluates that expression, “in line”, in place of the original macro application
- Simplified, multi-level backquote

Features / Utilities

- “Auto-load”
- Forget, setetop
- Macro Expander
- Tracer
- Debugger
- Many examples of language use, because ...
- Higher level forms are macros and functions

Features / “Bare Metal”

- Direct access to machine registers, from Lisp
- No cache, no virtual memory, just fast SRAM
- Breaks/Errors/Events interceptible by Lisp code
- Low latency GC, once top is “corraled”
- Add primitives at will, easily, in C
- Foreign function interface to libC, or other C

JIT Interpreter

- Lisp expressions expanded into VM instruction sequence sufficient to continue execution (basic block, decision point, etc.)
- VM instructions chosen to effect evaluation in the SECD machine model
- VM instructions patched in, replace Lisp code
- Simple expressions and macros are “inlined”
- Continue with VM, until next “uncracked” Lisp

SECD Virtual Machine

- S – stack (value value …)
- E – environment
  - (((x . 1) (y . 2)) ((z . 3) (h . 99)))
- C – control/code/command
  - List of VM instructions to execute
- D - “dump” - list/stack of S,E,C frames

SECD Virtual Machine

- Completely canonical SECD, but list surgery done where effects are equivalent
- TCO, naturally
- “Full” (is there any other kind ?) continuations
- ALL data on the heap
- () - list end, expression end VM instruction

Implementation – GC

- Cheney copying collector
- Reader and some primitives use other side
- Old “generation” is “eternal” top environment
- “Write barrier” is change in top environment
- Copy top, mark end, split the rest in two
- Copy the rest of the roots
- Don't have to collect top again until it changes
- Guard page, check between “basic blocks”

Implementation – Break/Errors

- Exceptions and errors create a value of a symbol, which is the error message
- Lisp code can specify a continuation to be applied in case of any error/exception
- ^C breaks are just the error “^C”
- Interrupts (will be) done similarly
- Breaks / interrupts can be deferred

Implementation – Backquote

- With one level, works just like any other
- But, each backquote observes every leading (left) unquote in the expression it is given, regardless of other backquotes inside the expression
- When nesting, add “ ,' “ as many times as necessary to defer to the right depth
- Smaller implementation, simpler rule to follow

```
`(global ,f (macro args
  `(loadapply ,file ,f ,',@args)))
```

Performance comparisons

- 30 times slower than C
- 3 times slower than Forth
- 3 times faster than Python
- Fact, Fib, Tak
- Clock for clock, 2x ? other not-so-JIT Lisps
- Comparable to 'FemtoLisp' (different kind of JIT)
- Slower than fully-compiled Lisps
- “Lispier”, more leverage, than Forth or Python

Because of Lisp ...

- Need more RAM, less ROM
- Don't care so much about other ecosystem support, language is the ecosystem
- Simple, uniform memory hierarchy preferred
- CPU ISA not a factor
- Digi-Key search
- … but Z80 / CP/M was a nice surprise

Because of System ...

- JIT to threaded VM code, not binary
- Simple flat pointers, simple heap
- 'forget' feature default
- Improved reader performance, symbol hash

Computer Science Education

- If kids must code …
- Law of primacy
- Distraction free
- Focus on essential ideas, not contingencies
- Don't saddle them with things to unlearn
- In the beginning, there were two choices
- One led to 50 years of learning the hard way

Besides Lisp ...

- CP/M running now
- Preliminary Nuttx port (thanks Greg !)
- Good for other things - there just isn't much to do, system resources mostly un-dedicated, system is not prematurely architect-ed
- MakerLisp is portable C, runs on Linux, and soon, on Nuttx
- Fast cheap hardware for straight embedded, too
- I don't care, I'll help no matter what you want to do

Demonstration / Q&A

- fact – cat, trace, debug
- + expanded
- Blinky
- shyard, oshyard (objects)
- Yes, you can have one if you'd like

CPU: $129.00 ($75 special Lisp/Functional Programming Group Mass Buy Offer)

I/O expansion: $89.00

USB: $70.00

VGA: $79.00

Enclosure: $99.00

System: $425.00

Prices will come down, soon

Evaluation Scheme

- 1. Constant ? “Quote” Value
- 2. Symbol ? Look up value of variable
- 3. List ?
  - a. special form ? "call/cc", "define", "if", "lambda", "macro", "progn", "quote", "setq"
  - b. Macro application ?
  - c. Function (primitive or abstraction) application
- Anything else is an error

Applying a Function

- Call site: empty stack, evaluate function object and arguments, then VM instruction “apply”
- Apply: primitive function ? just go
- Abstraction: recover the environment, bind the values on the stack to the parameters, extend environment with this new lexical level, set C to code body, continue

VM commands/instructions

- C_APPLY
- C_ARGC
- C_CONTINUE
- C_DEFINE
- C_END
- C_EVALC
- C_GET00
- C_GET10
- C_GET20
- C_GETB
- C_GETD
- C_GETL
- C_JUMPC
- C_JUMPM
- C_LAMBDA
- C_LOAD
- C_LOADC
- C_MACRO
- C_MAKECC
- C_QUOTE
- C_SELECT
- C_SETB
- C_SETD
- C_SETL