Microcoding Considered
As A Fine Art

May 1988

Bob Supnik

For Internal Use Only
If once a man indulges himself in microcoding, very soon he comes to think little of assembly coding; and from assembly coding he next comes to Fortran and Forth; and from that to terse comments and goto statements.

Contents:

- Microcoding.
- Microarchitecture.

SEG/AFL  Microcoding
• (Narrow) definition: Microcoding is the implementation of an instruction set interpreter on a low-level hardware engine.

• Goals (descending priority):
  – Accuracy.
  – Performance.
  – Schedule.
  – Space.
  – Maintainability.

• Non-goals:
  – Plasticity.
  – Modularity.
  – Aesthetics.
The Design Process

- Preparation - studies, prework.
- Comparative analysis - plagiarism no vice.
- Algorithm development - ways and means.
- Coding - implementation and documentation.
- Verification - functional, restrictive, dynamic.
- Optimization - squeezing the cycles.
Preparation

• Study the target ISP; know the SRM **cold**:
  – Complex instruction definitions.
  – Interactions between specifiers and execution.
  – Memory management.
  – Interrupts and exceptions.

• Study the micromachine:
  – Control and branching structure.
  – Parallel capabilities.
  – Serial restrictions.
  – Duplicate facilities.

• Understand performance tradeoffs in the architecture.

• Set goals and priorities: space vs speed, etc.

• Establish conventions for coding.

• Choose tools for development and verification.

*SEG/AFL*  

*Microcoding*
Comparative Analysis

“Plagiarism is the highest form of flattery.”

- Study past implementations:
  - Algorithms for complex instructions.
  - Space vs speed tradeoffs.
  - Features that worked especially well.
  - Features that were not used.
  - Utilization of parallelism.
  - Bugs that escaped pre-PG verification.

- Study contemporaneous implementations:
  - Study list is the same as given above.
  - Establish contact with other microcode projects.
  - Review others’ microcode as it is developed.
  - Invite others to review the code.
• Start with a baseline sketch, derived from:
  – Simple implementation of SRM description.
  – Direct translation of past implementation.
  – Direct translation of contemporary implementation.

• Critique by constraints:
  – Is decision tree compressed to minimum?

• Critique by comparison:
  – Does it meet performance goals?
  – Is it as fast, or faster, than past machines?
  – Is it as fast, or faster, than contemporary machines?

• Critique by dead space:
  – Are there NOPs?

• Examples: Rigel VFIELD, Rigel CSTRING.
• Start with register allocation.
  – Minimize (eliminate) data moves.
  – Maximize use of common routines and common exits.

• NOPs are **verboten**!
  – Shorten the decision tree.
  – Push calculations up the tree.
  – Use cases instead of compares.
  – Add additional functions to fill slots.

• Think parallel.
  – Multiple tests from one calculation.
  – Multiple consequences of one action.
  – Multiple interpretations of one result.
  – Multiple actions on one source.
• Think hardware.
  – What hardware features are not being used, and could be eliminated?
  – What microcode routines are not meeting goal, and could use additional hardware support?

• Document, document, document!
  – Introductions to entry points, algorithms, etc.
  – Specialized comments on microcode restrictions.
  – State description on every continuation page.
  – Cycle counts for complex decision trees.

• Code review.
  – When code is complete, or even before.
  – Starting point is accuracy.
  – Next issue is performance.
  – Next issue is space.
  – Subordinate points: reusability, allocation plasticity.
• Rigel VFIELD:

  \[ \text{[SC]} \leftarrow \text{000000}[32.] - \text{[MD.T2]}, \text{LONG} \]

  - Tests size > 32.
  - Using SC case, tests size = 0.

• Rigel VFIELD:

  \[ \text{[SC]} \leftarrow \text{[MD.T0]} - \text{000000}[32.], \text{LONG} \]

  - Tests position > 31.
  - Loads position to SC (SC operates mod 32).

• Rigel QUEUE:

  \[ \text{MEM(}V\text{A)}\&, \text{[SC]} \leftarrow \text{[MD.T0]} - \text{[SC]}, \text{LONG} \]

  - Calculates result.
  - Saves result for later use.
  - Writes result to memory.
• Functional verification:
  – HCORE - for initial debug.
  – AXE - for thorough coverage; use latest version, always!
  – MAXE - for pipeline interactions.
  – SEGUE - for pipeline interactions.
  – ELN - a reasonably short OS test.
  – VMS - the ultimate test.

• Restrictive verification:
  – ARCS - microcode restrictions check.

• Dynamic verification:
  – Check every error and exception path!
  – Interrupts (regular and passive release).
  – DMA and invalidates.
  – Coincident transactions (prefetch+, invalidate +, error+), etc.

SEG/AFL

Microcoding
Optimization

- Microcode, like bread dough, needs to “rest” a while before being worked again.
- After initial verification, take a break.
- Then look for “peephole optimizations”:
  - Eliminate NOPs by restructuring, filling, etc.
  - Eliminate STALLs by scheduling parallel work.
  - Eliminate MOVEs by revised register allocation.
  - Eliminate COMPAREs by cases.
  - Eliminate duplicate actions.
  - Eliminate words by finding common sequences.
  - Eliminate words by finding one line subroutines.
  - Eliminate allocation bottlenecks by alignlist strength reduction.
- The number of changes needed to free up a word or cycle may be enormous!

SEG/AFL

Microcoding
Optimization Examples

- **RIGEL MISC**: Eliminated two NOPs in INDEX by:
  - Adding functionality (size = 1 test).
  - Pushing calculation (subscript + size) up the tree.
  - Casing into subroutine instead of calling.
  - Saved multiple cycles, words too!

- **RIGEL CALLRET**: Eliminated cycle in CALLs by:
  - Making one shift serve two purposes (align mask for call frame, align mask for casing).

- **RIGEL CALLRET**: Eliminated STALL in RET by:
  - Placing more work under LOAD PC shadow.
  - Rewriting as needed to free up work for shadow.

- **RIGEL MULDIV**: Eliminated register copy in DIVIDE by:
  - Noticing “useless” move on error path.
  - Reallocating registers to eliminate move.
Optimization Examples

- **RIGEL OPSYS**: Eliminated COMPAREs in CHMx by:
  - Implementing full case tree for opcode vs current mode.
  - Saved compare, extra data move.
  - Eliminated “wrong choice” path.
  - Saved 2+ cycles, no extra words.

- **RIGEL MISC**: Eliminated duplicate read in POPR:
  - Prologue reads end of stack frame to test accessibility.
  - Main loop unwinds mask bits \(<11:0>\).
  - Epilogue unwinds mask bits \(<14:12>\).
  - Reusing longword saves 3 words, can save cycles.

- **RIGEL INTLOG**: Eliminated words, cycles in ASHQ by:
  - Noticing right shift case had extra cycle due to conflict between condition code order requirements and shift order requirements.
  - Reused MOVQ storage routine to set condition codes.
  - Allowed shift to be done in optimized order.

*SEG/AFL* Microcoding
Random Thoughts

- Ordinary programming (systems or application) and microcoding are very different.
  - An ordinary program is implemented once, with a view towards long term maintenance and modification.
  - Microcode may be implemented many times, but once finished is complete.
  - Microcode is hacking at its best: the last refuge of the assembly language fanatic.
- The worst enemy of good microcode is NIH.
  - Beg, borrow, and steal good ideas from others.
  - Use others to review and critique code.
- Microcode demands optimization, and optimization demands multiple passes.
  - Multiple passes, dispersed in time, by the same person.
  - Multiple reviewers, at the same time.

SEG/AFL

Microcoding
• (Narrow) definition: Microarchitecture is the process of defining a low-level hardware engine for a microcoded processor implementation.

• Goals (descending priority):
  – Implementation feasibility.
  – Performance.
  – Implementation complexity.
  – Schedule.
  – Implementation cost.

• Non-goals:
  – Extensibility.
  – Reusability.
  – Aesthetics.
The Design Process

- Preparation - studies, prework.
- Comparative analysis - the state of the art.
- Microword development - trying out ideas.
- Tradeoffs - hardware cost and microcode cost.
- Formal definition - the final result.
• Study the SRM; how does the architecture impact hardware?
  – Basic data path requirements (eg, GPRs, working registers, ALU, shifter, condition codes, RLOG, etc).
  –Specifier decomposition.
  – Memory management.
  – Interrupts and exceptions.

• Study the “fine print” backbreakers.
  – Unaligned memory references.
  – Conflicting specifier usage (eg, (R),(R)+).
  – Double write specifiers.
  – Implicit specifiers.

• Understand reliability and recoverability requirements for target systems.

• Understand performance tradeoffs in the architecture.

• Set goals and priorities: cycle time, cpi, etc.

*SEG/AFL Microcoding*
Comparative Analysis

- Study past implementations:
  - Data path facilities.
  - Instruction parsing facilities.
  - Memory management facilities.
  - BIU structure.
  - Cache and memory structures.
  - Microcode structure (horizontal vs vertical, serial vs parallel).
  - Handling of architectural nasties.
  - Performance return from individual features.

- Study contemporaneous implementations:
  - Study list is the same as given above.

- Study hypothetical implementations:
  - Cache and memory subsystem configurations.
  - Theoretical limits on implementation efficiency.

SEG/AFL

Microcoding
Microword Development

• Goal: first cut at a microword (E Box structure).

• Starting point: select a theme.
  – MicroVAX - get it to fit (microword existed)!
  – CVAX (first theme) - minimize microword count.
  – CVAX (final theme) - minimize logic.
  – Rigel - allow parallel E Box, BIU operations.
  – NVAX - minimize cpi of complex instructions.

• Select a sequencing style.

• Code key routines in “free form” microcode.
  – Specifiers.
    – Integer/logical, control, field, procedure call.

• Derive minimum set of fields and functions per field.

• Define initial microword.
• Initial CVAX goal was minimum number of microwords:

• Final CVAX goal was minimum amount of logic while maintaining narrow microword:
  – MicroVAX’ 9 data path formats reduced to 5.
  – MicroVAX’ 32 destination selects reduced to 4.
  – MicroVAX’ 32 branch conditions reduced to 16.
  – MicroVAX’ 16 way cases reduced to 8, corresponding to hardware organization of CS ROM.
  – MicroVAX’ 9 literal formats reduced to 4.
  – MicroVAX’ 8 CC recipes reduced to 4.
  – MicroVAX’ 8 state flags reduced to 6.
  – MicroVAX’ signed offset addr changed to page mode.
  – MicroVAX’ conditional branches eliminated.
  – Fields common to all formats always in same place.
  – Memory request field horizontally encoded.
  – Special control functions horizontally encoded.
  – Duplicate data path functions eliminated.
  – Microword grew from 39b to 41b.
Rigel goal was to augment CVAX with parallel MRQ and ALU functions and to further simplify decoding:

- Combined MRQ/ALU format to support read and run.
- Also supports calculate and write.
- Destination select replaced by explicit destination field for simpler decoding.
- Microword grew from 41b to 50b, since ROM width was no longer much of a performance issue.
- Wider format allowed further simplification of shift format.

The main challenge of Rigel was not in the data path but in the sequencing.

- Micropipeline implied longer branch latencies.
- In particular, 1 cycle ALU latency grew to 3 cycles.
- Could the microcode cope with the extra latency?
- Feasibility proven by probe coding.
• NVAX is a macropipelined machine.

• The largest irreducible component of cpi is complex instructions which require the I Box to shut down during execution.

• Therefore, the goal of the (E Box) microarchitecture must be minimized cpi.

• The Nautilus and Aquarius microcode are directly applicable.

• Possible I/O facilities:
  – Automatic compaction of related writes into quadwords.
  – I/O operation every cycle at external pins.
  – Separate I and D (I/D) caches.

*SEG/AFL* Microcoding
• Possible microcode facilities:
  – Tailored ALU operations (eg, sign extend).
  – Tailored shift operations (eg, sign extend).
  – Parallel MRQ, ALU, SHF operations for maximum parallelism.
  – Tailored register operations (eg, byte writable PSL).
  – Tailored branch conditions (eg, case on all interesting CALLx mask bits).
  – Special function units for complex instructions (eg, mask unit, population counter, REI validator).
  – Shortened microbranch latency.
  – Microbranch tests at ALU/SHF input as well as output.
Ultimately, every microarchitectural feature must be justified by an SRM constraint or by a performance payback.

- All VLSI uVAXen have featured:
  - Hardware implemented unaligned I/O.
  - Narrow, horizontally encoded microwords.
  - <2k word control store limit.
  - 32b right funnel shifter.
  - Multifunction SC register.

- Some CVAX tradeoffs:
  - Opcode dependent ALU functions rejected.
  - Per-instruction register optimization rejected.
  - SC casing limited to bits <5:0>.
  - SISR partially implemented in hardware.
  - CC map select moved from IPLA to microword.
  - Branch logic enhanced to support loop branches.
  - Branches implemented via microtrap.
  - 780-like I Box replaced by 8800-like I Box.
  - VA on B Bus to speed up TB miss flows.
• Some Rigel tradeoffs:
  – Per-instruction register optimization added.
  – SC casing broaded to bits $<11:0>$.
  – Population counter added.
  – REI, mask validator rejected.
  – SISR fully implemented in hardware.
  – Edge triggered latches reset by microcode.
  – Memory management length checks done in E Box.
  – No static ALU condition codes.
  – RLOG decoding done in hardware.

The list of decisions is endless.
Formalization

- The tentative microword definition is checked for hardware implementation feasibility.
  - Ease of decoding and execution.
  - Minimization of hardware maintained state.
  - Effect on cycle time.

- The sketch feasibility microcode is fleshed out to be a full scale trial implementation.
  - Flushes out missing features for obscure cases.
  - Ensures adequate facilities for exceptions.

- The trial implementation is used to drive a performance model of the system, to validate performance goals.

- At the conclusion of performance modelling, the microcode definition is tentatively frozen.

- **But**, there is always room for inventiveness, or catastrophe, to strike.
Concluding Thoughts

- Tbd.