Introduction

* NIGE Machine presented at EuroFORTH 2012
* Improvements identified
  + Native FAT file system on SSD card, now implemented
  + Improved SRAM memory architecture from 8 bit to 32 bit width, this paper
  + Port to more advanced FPGA board, testing underway
* Widen instruction fetch whilst preserving non-aligned memory access
* Developed
* Results, speed increase of 25% but at the expense of a longer logic and potentially slower clock speeds
* Either the byte width or long width memory versions could be most suitable for future development

Problem description

* Original issue
  + Specifications (and reasons why)
    - Byte width instructions and non-aligned memory access (code density)
    - Single cycle instruction throughput (performance)
    - Deterministic execution (embedded)
  + Consequences of original memory design (byte access)
    - Byte width instructions and non-aligned access natural
    - Load literals not single cycle, four cycles to implement a branch
* Why this is important for a FORTH machine
  + Frequent subroutine calls
  + Byte-sized instruction format means that instructions can be placed at any position wrt a word boundary
  + Frequent subroutine calls in FORTH meant that reducing the number of cycles to read an address and branch to it offered attractive gains

Why this is not straightforward

* RAM, including FPGA block RAM, organized as width \* depth
* Not possible to cross a word boundary with a read or write
* Boundary condition could break deterministic execution
* FIG: explain the above

Outline of solution

* Dual ported SRAM and SRAM controller to synthesize longword misaligned memory
* Incorporate variable instruction sizes into CPU so that multi-byte instructions can execute in a single cycle
* Create a new, single cycle, JSL instruction

SRAM controller design update

* Using FPGA dual ported RAM, possible to create non-aligned single cycle access
* Control line to specify B/W/L size
* Same system can also deliver (addr+1) for literal loads in a single cycle
* Writes now require 2 cycles, since for #.W and #.L first to read, then to write
* FIG: block diagram of inputs and outputs connected to SRAM
* FIG: examples of how the reads are synthesized

Control unit design update

* Variable instruction length
* Four-stage pipeline [TABLE]
  + Read instruction size
  + Fetch opcode
  + Decode and execute
  + Save
* JSL instruction

Datapath update

* Load literal vs. memory load

Results

How the results were obtained

* Cycles per instruction [TABLE] , and why like that
* Instruction frequency usage [TABLE]
* Benchmarks speed up [TABLE]
* Increased logic levels and maximum clock speed [TABLE]

Discussion

* Significant improvements obtained but at a cost of increased logic
* Impact on maximum clock speed
* Alternative approaches
  + Further lengthen pipeline (actually byte-by-byte instruction read can be considered a form of benchmarking
  + Force JSL and sore/load to align on longword boundaries

Conclusion

* Successfully implemented longword memory access
* Speed enhancement is a subtle goal
* Trade-off decision as to which design to adopt