ESE535: Electronic Design Automation

Day 23: April 27, 2008

Processor Verification

Today

- Specification/Implementation
- Abstraction Functions
- Correctness Condition
- Verification
- Self-Consistency

Specification

- Abstract from Implementation
- Describes observable/correct behavior

Implementation

- Some particular embodiment
- Should have **same** observable behavior
  - Same with respect to **important** behavior
- Includes many more details than spec.
  - How performed
  - Auxiliary/intermediate state

Important Behavior

- Same output sequence for input sequence
  - Same output after some time?
- Timing?
  - Number of clock cycles to/between results?
  - Timing w/in bounds?
- Ordering?

Abstraction Function

- Map from implementation state to specification state
  - Use to reason about implementation correctness
  - Want to guarantee: AF(Fi(q,i))=Fs(AF(q),i)

$$\begin{tikzcd}
\bullet \arrow[r, "AF"] \arrow[d, "Fi"'] & \bullet \arrow[d, "Fs"] \\
\bullet \arrow[r, "AF"'] & \bullet
\end{tikzcd}$$

Familiar Example

- Memory Systems
  - Specification:
    - W(A,D)
    - R(A)→D from last D written to this address
  - Specification state: contents of memory
  - Implementation:
    - Multiple caches, VM, pipelined, Write Buffers…
  - Implementation state: much richer…

Memory AF

- Maps from
  - State of caches/WB/etc.
- To
  - Abstract state of memory
- Guarantee AF(Fi(q,I))==Fs(AF(q),I)
  - Guarantee change to state always represents the correct thing

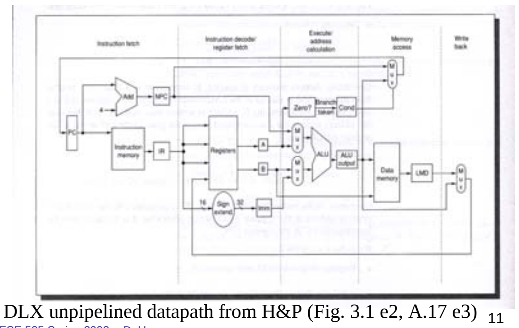
Abstract Timing

- For computer memory system
  - Cycle-by-cycle timing not part of specification
  - Must abstract out
- Solution:
  - Way of saying "no response"
    - Saying "skip this cycle"
    - Marking data presence
      - (tagged data presence pattern)

Filter to Abstract Timing

- Filter input/output sequence
- Os(in)→out
- FilterStall(Impl_{in}) = in
- FilterStall(Impl_{out}) = out
- Forall sequences Impl_{in}
  - FilterStall(Oi(Impl_{in})) = Os(FilterStall(Impl_{in}))

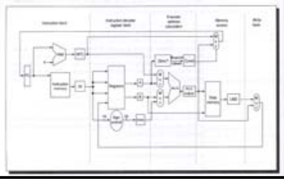
DLX Datapath



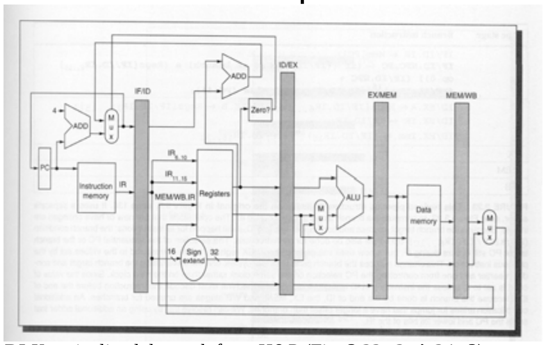
DLX unpipelined datapath from H&P (Fig. 3.1 e2, A.17 e3)

Processors

- Pipeline is big difference between specification state and implementation state.
- Specification State:
  - Register contents (incl. PC)
  - Memory contents

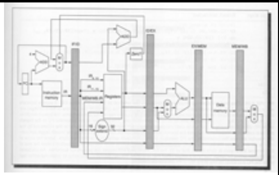


Revised Pipeline



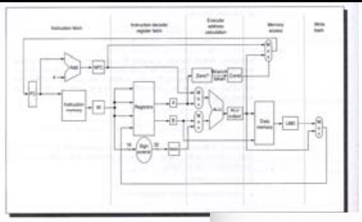
DLX repipelined datapath from H&P (Fig. 3.22 e2, A.24 e3)

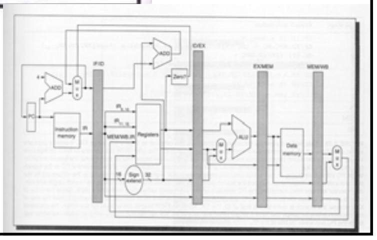
Processors



- Pipeline is big difference between specification state and implementation state.
- Specification State:
  - Register contents (incl. PC)
  - Memory contents
- Implementation State:
  - + Instruction in pipeline
  - + Lots of bits
    - Many more states
    - State-space explosion to track

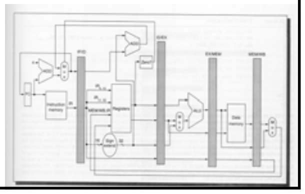
Compare





Observation

- After flushing pipeline,
  - Reduce implementation state to specification state
- Can flush pipeline with series of NOPs or stall cycles



Pipelined Processor Correctness

- w = input sequence
- w_f = flush sequence
  - Enough NOPs to flush pipeline state
- Forall states q and prefix w
  - Fi(q,w w_f)→Fs(q,w w_f)
  - Fi(q,w w_f)→Fs(q,w)
- FSM observation
  - Finite state in pipeline
  - only need to consider finite w

Pipeline Correspondence

$$\begin{tikzcd}
\boxed{\text{Old Impl State}} \arrow[r, "F_{\mathrm{Impl}}(\cdot, I_{\mathrm{Stall}})"] \arrow[d, "F_{\mathrm{Impl}}(\cdot, I)"] & \cdots \arrow[r, "F_{\mathrm{Impl}}(\cdot, I_{\mathrm{Stall}})"] & \boxed{\text{Flushed Old Impl State}} \arrow[r, "proj"] & \boxed{\text{Old Spec State}} \arrow[d, "F_{\mathrm{Spec}}(\cdot, I)"'] \\
\boxed{\text{New Impl State}} \arrow[r, "F_{\mathrm{Impl}}(\cdot, I_{\mathrm{Stall}})"'] & \cdots \arrow[r, "F_{\mathrm{Impl}}(\cdot, I_{\mathrm{Stall}})"'] & \boxed{\text{Flushed New Impl State}} \arrow[r, "proj"'] & \boxed{\text{New Spec State}}
\end{tikzcd}$$

[Burch+Dill, CAV'94]

Equivalence

- Now have a logical condition for equivalence
- Need to show that it holds
  - Is a Tautology
- Or find a counter example

Ideas

- Extract Transition Function
- Segregate datapath
- Symbolic simulation on variables
  - For q, w's
- Case splitting search
  - Implication pruning

Extract Transition Function

- From HDL
- Similar to what we saw for FSMs

Segregate Datapath

- Big state blowup is in size of datapath
  - Represent data symbolically/abstractly
    - Independent of bitwidth
  - **Not** verify datapath/ALU functions as part of this
    - Can verify ALU logic separately using combinational verification techniques
    - Abstract/uninterpreted functions for datapath

Burch&Dill Logic

- Quantifier-free
- Uninterpreted functions (datapath)
- Predicates with
  - Equality
  - Propositional connectives

B&D Logic

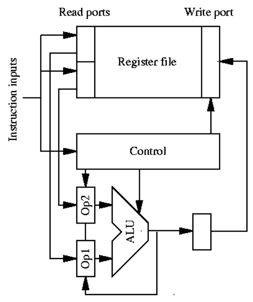
- Formula = **ite**(formula, formula, formula)
  | (term=term)
  | psym(term,…term)
  | pvar | **true** | **false**
- Term = **ite(**formula,term,term)
  | fsym(term,…term)
  | tvar

Sample

- Regfile:
  - (ite stall

```
(ite stall
     regfile
     (write regfile
            dest
            (alu op
                 (read regfile src1)
                 (read regfile src2))))
```

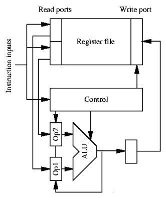
Sample Pipeline



Example Logic

- arg1:

```
(ite (or bubble-ex
         (not (= src1 dest-ex)))
     (read
        (ite bubble-wb
             regfile
             (write regfile dest-wb result))
        src1)
     (alu op-ex arg1 arg2))
```



Symbolic Simulation

- Create logical expressions for outputs/state
  - Taking initial state/inputs as variables

Case Splitting Search

- Satisfiability Problem
- Pick an unresolved variable
- Branch on true and false
- Push implications
- Bottom out at consistent specification
- Exit on contradiction
- Pragmatic: use memoization to reuse work

Review: What have we done?

- Reduced to simpler problem
  - Simple, clean specification
- Abstract Simulation
  - Explore all possible instruction sequences
- Abstracted the simulation
  - Focus on control
  - Divide and Conquer: control vs. arithmetic
- Used Satisfiability for reachability in search in abstract simulation

Achievable

- Burch&Dill: Verify 5-stage pipeline DLX
  - 1 minute in 1994

Self Consistency

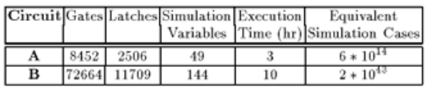
Self-Consistency

- Compare same implementation in two different modes of operation
  - (which should not affect result)
- Compare pipelined processor
  - To self w/ NOPs separating instructions
    - So only one instruction in pipeline at a time

Self-Consistency

- w = instruction sequence
- S(w) = w with no-ops
- Show: Forall q, w
  - F(q,w) = F(q,S(w))

Sample Result

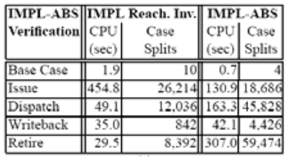


Circuit	Gates	Latches	Simulation Variables	Execution Time (hr)	Equivalent Simulation Cases
A	8452	2506	49	3	$6 * 10^{14}$
B	72664	11709	144	10	$2 * 10^{43}$

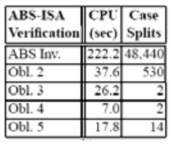
Table 1. Self-consistency checking results.

[Jones, Seger, Dill/FMCAD 1996]
n.b. Jones&Seger at Intel

Sample Result



IMPL-ABS Verification	IMPL Reach. Inv.		IMPL-ABS	
	CPU (sec)	Case Splits	CPU (sec)	Case Splits
Base Case	1.9	10	0.7	4
Issue	454.8	26,214	130.9	18,686
Dispatch	49.1	12,036	163.3	45,828
Writeback	35.0	842	42.1	4,426
Retire	29.5	8,392	307.0	59,474



ABS-ISA Verification	CPU (sec)	Case Splits
ABS Inv.	222.2	48,440
Obl. 2	37.6	530
Obl. 3	26.2	2
Obl. 4	7.0	2
Obl. 5	17.8	14

Verification running on P2-200MHz

[Skakkebæk, Jones, and Dill / CAV 1998]

Key Idea

- Implementation State reduces to Specification state after finite series of operations
- Abstract datapath to avoid dependence on bitwidth

Admin

- Last Class
- Final office hours T4pm
- Assignment 7 due May 13th
- I did make it clear you cannot pass class without completing the programming assignments (3,4,5).

Big Ideas

- Proving Invariants
- Divide and Conquer
- Exploit structure