

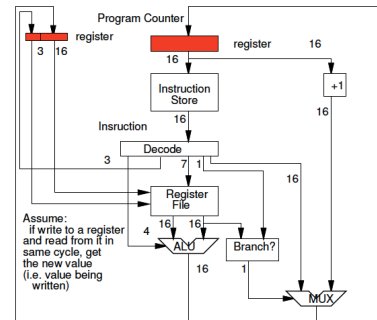
# ESE535: Electronic Design Automation

Day 26: April 25, 2011  
Processor Verification



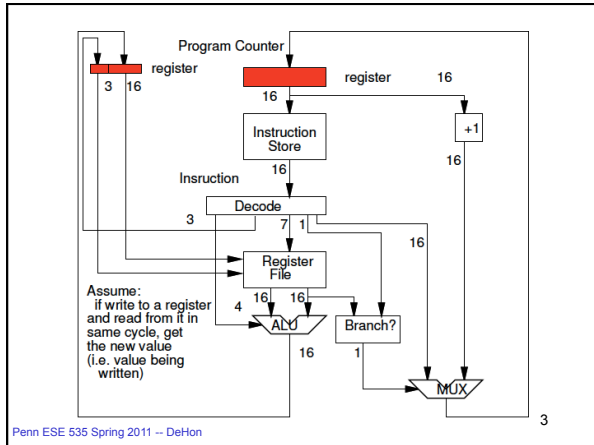
Penn ESE 535 Spring 2011 -- DeHon

## Can we pipeline?



Penn ESE 535 Spring 2011 -- DeHon

2

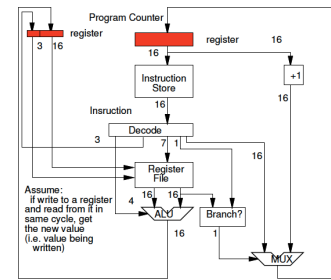


Penn ESE 535 Spring 2011 -- DeHon

3

## Pipelining: ALU-RF Path

- Only a problem when **next** instruction depends on value written by immediately previous instruction
- ADD R3←R1+R2
- ADD R4←R2+R4
- ADD R5←R4+R3

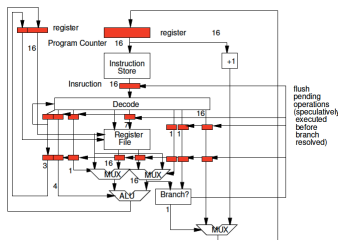


Penn ESE 535 Spring 2011 -- DeHon

4

## ALU-RF Path

- Only a problem when **next** instruction depends on value written by immediately previous instruction
- Solve with Bypass

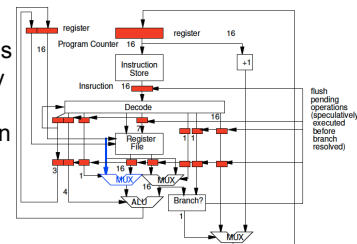


Penn ESE 535 Spring 2011 -- DeHon

5

## ALU-RF Path

- Only a problem when **next** instruction depends on value written by immediately previous instruction
- Solve with Bypass

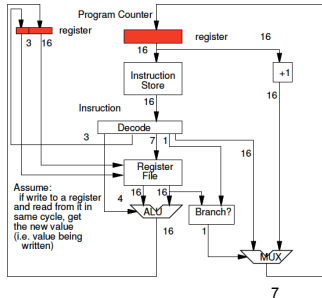


Penn ESE 535 Spring 2011 -- DeHon

6

## Branch Path

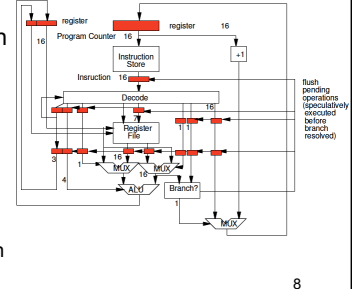
- Only a problem when the instruction is a taken branch



Penn ESE 535 Spring 2011 -- DeHon

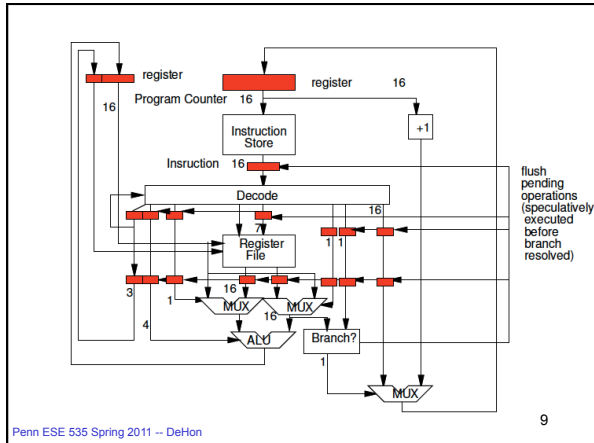
## Branch Path

- Only a problem when the instruction is a taken branch
- Solve by
  - Speculating is not a taken branch
  - Preventing the speculative instruction from affecting state when branch occurs



Penn ESE 535 Spring 2011 -- DeHon

8

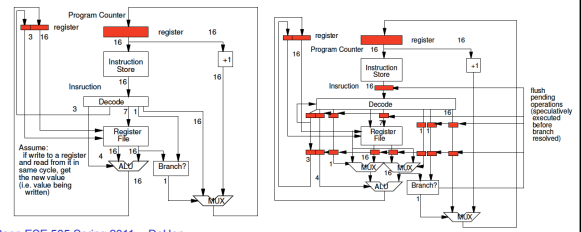


Penn ESE 535 Spring 2011 -- DeHon

9

## Example

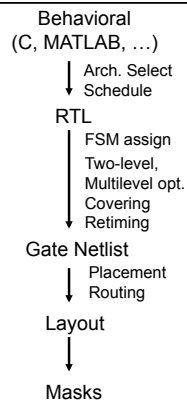
- Different implementations for same specification



Penn ESE 535 Spring 2011 -- DeHon

## Today

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



Penn ESE 535 Spring 2011 -- DeHon

11

## Specification

- Abstract from Implementation
- Describes observable/correct behavior

Penn ESE 535 Spring 2011 -- DeHon

12

## 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

## Unimportant Behavior?

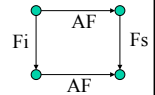
- What behaviors might be unimportant?

## “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?

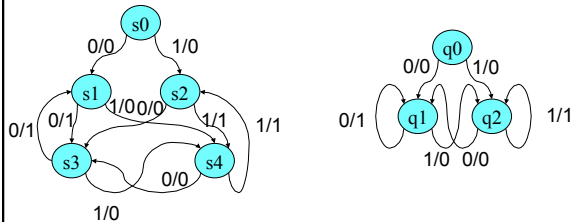
## Abstraction Function

- Map from implementation state to specification state
  - Use to reason about implementation correctness
  - Want to guarantee:  $AF(Fi(q,i))=Fs(AF(q),i)$ 
    - Similar to saying the composite state machines always agree on output (state)
      - ...but have more general notion of outputs and timing

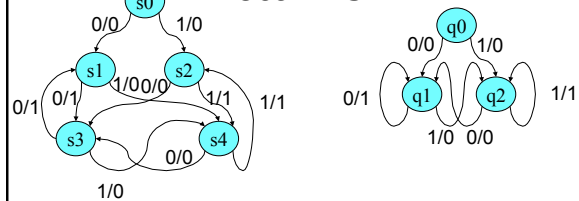


## Recall FSM

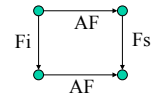
- Equivalent FSMs with different number of states



## Recall FSM



- Maybe right is specification
- $AF(s1)=q1$ ,  $AF(s3)=q1$
- $AF(s2)=q2$ ,  $AF(s4)=q2$
- $AF(s0)=q0$



## Familiar Example

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

Penn ESE 535 Spring 2011 -- DeHon

19

## Memory AF

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

Penn ESE 535 Spring 2011 -- DeHon

20

## Memory: L1, writeback

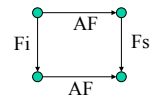
- Memory with L1 cache
  - L1 cache is extra state
    - Another L1.capacity words of data
  - Check L1 cache first for data on read
  - Miss  $\rightarrow$  load into cache
  - Writes update mapping for address in L1
  - When address evicted from L1
    - write-back to main memory

Penn ESE 535 Spring 2011 -- DeHon

21

## Memory: L1, writeback

- Specification State:
  - one memory with addr:data mappings
  - $M(a) = MM[a]$
- L1 writeback cache implementation
  - $AF(L1+M)$ : for all a
    - If a in L1
    - $M(a) = L1[a]$
    - else
    - $M(a) = MM[a]$

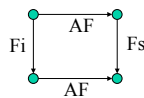


Penn ESE 535 Spring 2011 -- DeHon

22

## Memory: L1, writeback

- Specification State:
  - one memory with addr:data mappings
  - $M(a) = MM[a]$
- What are several (different) implementation states that map to same specification state?
  - Concrete:  $M(0x100C) = 0xBEC1$



Penn ESE 535 Spring 2011 -- DeHon

23

## 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)
    - Example: stall while fetch data into L1 cache

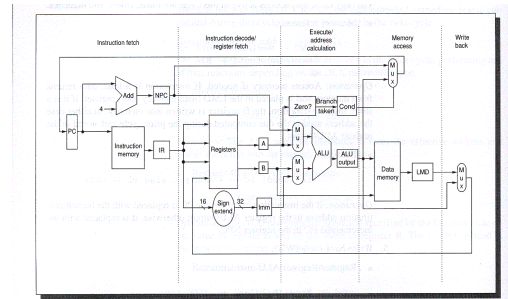
Penn ESE 535 Spring 2011 -- DeHon

24

## Filter to Abstract Timing

- Filter input/output sequence
- View computation as:  $O_s(in) \rightarrow out$
- $FilterStall(Impl_{in}) = in$
- $FilterStall(Impl_{out}) = out$
- For all sequences  $Impl_{in}$ 
  - $FilterStall(O_i(Impl_{in})) = O_s(FilterStall(Impl_{in}))$

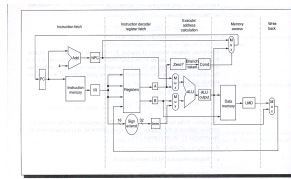
## DLX Datapath



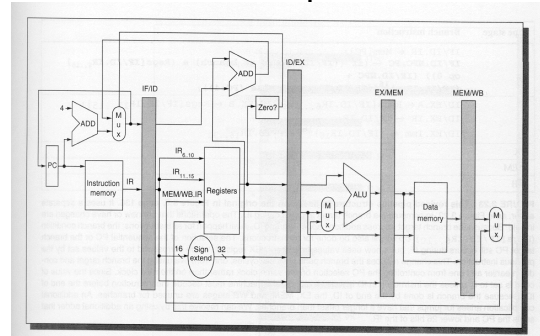
DLX unpipelined datapath from H&P (Fig. 3.1 e2, A.17 e3)

## Processors

- Pipeline is big difference between specification state and implementation state.
- What is specification state?

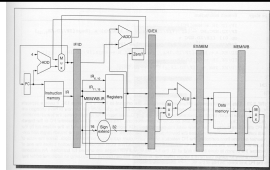


## Revised Pipeline



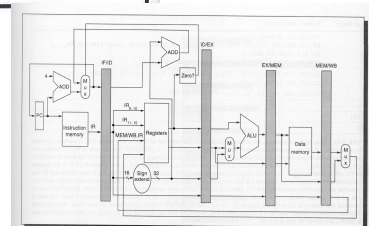
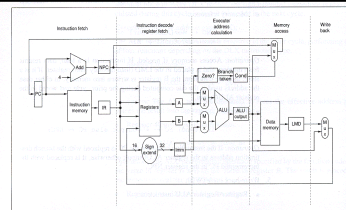
DLX repipelined datapath from H&P (Fig. 3.22 e2, A.24 e3)

## Processors



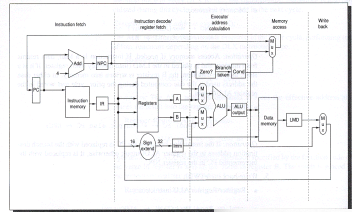
- Pipeline is big difference between specification state and implementation state.
- Specification State:
  - PC, RF, Data Memory
- Implementation State:
  - + Instruction in pipeline
  - + Lots of bits
    - Many more states
    - State-space explosion to track

## Compare



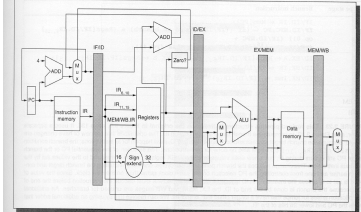
## Return to L1, writeback

- How does main memory state relate to specification state after an L1 cache flush?
  - L1 cache flush = force writeback on all entries of L1



## Compare

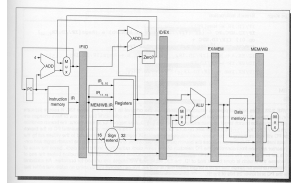
How make the shared state the same?



Penn ESE 535 Spring 2011 -- DeHon

## Observation

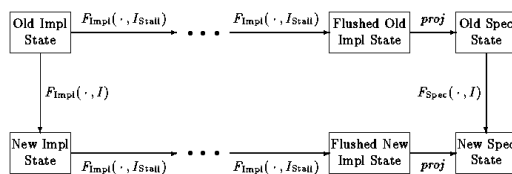
- After flushing pipeline,
  - Reduce implementation state to specification state (RF, PC, Data Mem)
- Can flush pipeline with series of NOOPs or stall cycles



## Pipelined Processor Correctness

- $w$  = input sequence
- $w_f$  = flush sequence
  - Enough NOOPs to flush pipeline state
- For all states  $q$  and prefix  $w$ 
  - $F_i(q, w, w_f) \rightarrow F_s(q, w, w_f)$
  - $F_i(q, w, w_f) \rightarrow F_s(q, w)$
- FSM observation
  - Finite state in pipeline
  - only need to consider finite  $w$

## Pipeline Correspondence



[Burch+Dill, CAV'94]

## Equivalence

- Now have a logical condition for equivalence
- Need to show that it always 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
  - Generalization of SAT
  - Uses 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

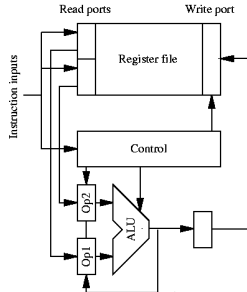
## B&D Logic

- 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  
regfile  
(write regfile  
dest  
(alu op  
(read regfile src1)  
(read regfile src2))))

## Sample Pipeline

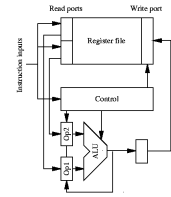


Penn ESE 535 Spring 2011 -- DeHon

43

## 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))



Penn ESE 535 Spring 2011 -- DeHon

44

## Symbolic Simulation

- Create logical expressions for outputs/ state
  - Taking initial state/inputs as variables
- E.g. (ALU op2 (ALU op1 rf-init1 rf-init2) rf-init3)

Penn ESE 535 Spring 2011 -- DeHon

45

## Case Splitting Search

- Satisfiability Problem
- Pick an unresolved variable
  - (= src1 dest-ex)
  - (= 0 (ALU op2 (ALU op1 rf-init1 rf-init2) rf-init3))

Penn ESE 535 Spring 2011 -- DeHon

46

## 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

Penn ESE 535 Spring 2011 -- DeHon

47

## 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

Penn ESE 535 Spring 2011 -- DeHon

48



## Achievable

- Burch&Dill: Verify 5-stage pipeline DLX
  - 1 minute in 1994
    - On a 40MHz R3400 processor
- Modern machines 30+ pipeline stages
  - ...and many other implementation embellishments

## Self Consistency

## Self-Consistency

- Compare same implementation in two different modes of operation
  - (which should not affect result)
- Examples of different modes of operation that should behave the same?

## Self-Consistency

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

## Self-Consistency

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

## Sample Result

- A – stream processor
- B – multithread pipeline

Circuit	Gates	Latches	Simulation Variables	Execution Time (hr)	Equivalent Simulation Cases
A	8452	2506	49	3	$6 * 10^{14}$
B	72664	11709	144	10	$2 * 10^{33}$

Table 1. Self-consistency checking results.

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

## Sample Result: OoO processor

IMPL-ABS Verification	IMPL Reach. Inv.		IMPL-ABS		ABS-ISA Verification	CPU (sec)	Case Splits
	CPU (sec)	Case Splits	CPU (sec)	Case Splits			
Base Case	1.9	10	0.7	4	ABS Inv.	222.2	48,440
Issue	454.8	26,214	130.9	18,686	Obl. 2	37.6	530
Dispatch	49.1	12,036	163.3	45,828	Obl. 3	26.2	2
Writeback	35.0	842	42.1	4,426	Obl. 4	7.0	2
Retire	29.5	8,392	307.0	59,474	Obl. 5	17.8	14

Verification running on P2-200MHz

[Skakkebæk, Jones, and Dill / CAV 1998,  
Formal Methods in System Design v20, p139, 2002]

Penn ESE 535 Spring 2011 -- DeHon

55

## Key Idea Summary

- Implementation state reduces to Specification state after finite series of operations
- Abstract datapath to avoid dependence on bitwidth
- Abstract simulation (reachability)
  - Show same outputs for any input sequence
- State→state transform
  - Can reason about finite sequence of steps

Penn ESE 535 Spring 2011 -- DeHon

56

## Admin

- Last Class
- Assignment 8 out
  - due May 9<sup>th</sup> (noon)
  - Late assignments will **not** receive partial credit
  - André traveling May 1—6
    - Ask clarifying questions before May 1
- Normal office hours Tuesday (tomorrow)
  - None on May 3rd
- Course evaluations online

Penn ESE 535 Spring 2011 -- DeHon

57

## Big Ideas

- Proving Invariants
- Divide and Conquer
- Exploit Structure

Penn ESE 535 Spring 2011 -- DeHon

58