CS/ECE 552: Pipeline Hazards

Prof. Matthew D. Sinclair

Lecture notes based in part on slides created by Mark Hill, Mikko Lipasti, David Wood, Guri Sohi, John Shen and Jim Smith

Pipeline Hazards

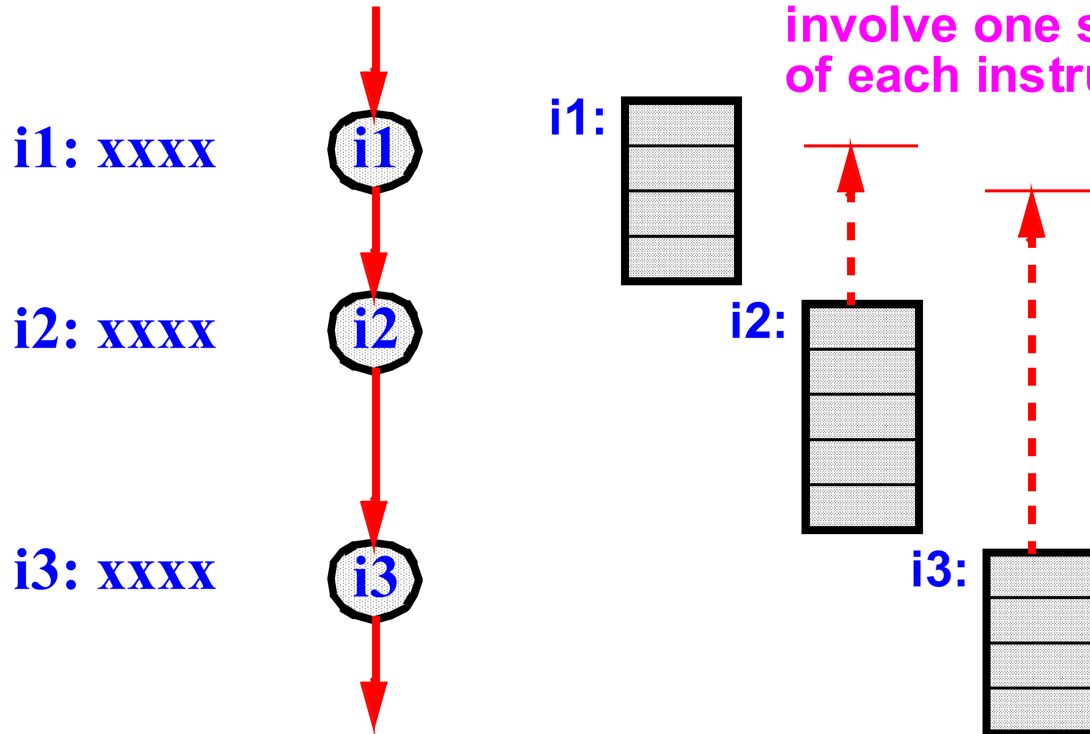
- Forecast
  - Program Dependences
  - Data Hazards
    - Stalls
    - Forwarding
  - Control Hazards
    - Stalls
    - Speculation
  - Exceptions

Sequential Execution Model

- MIPS ISA requires the appearance of *sequential execution*
  - *Precise exceptions*
  - True of most general-purpose ISAs
  - Hardware's goal: maintain this illusion
    - Execute things concurrently under the hood
    - Use bookkeeping to keep track of sequential order
    - If something bad happens (e.g., exception): utilize bookkeeping to restore everything to sequential order

Program Dependences

A true dependence between two instructions may only involve one subcomputation of each instruction.



The implied sequential precedences are an overspecification. It is sufficient but not necessary to ensure program correctness.

Program Data Dependences

- True dependence (RAW)
  - j cannot execute until i produces its result

$$D(i) \cap R(j) \neq \phi$$

- Anti-dependence (WAR)
  - j cannot write its result until i has read its sources

$$R(i) \cap D(j) \neq \phi$$

- Output dependence (WAW)
  - j cannot write its result until i has written its result

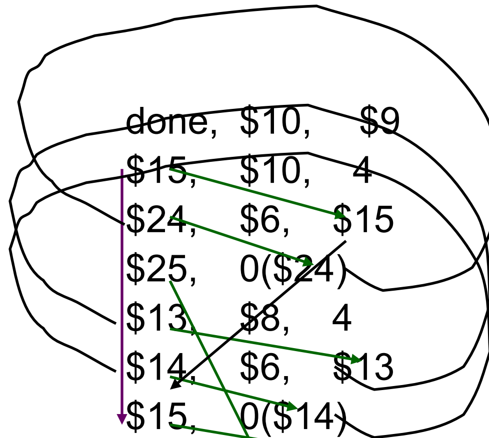
$$D(i) \cap D(j) \neq \phi$$

Control Dependences

- Conditional branches
  - Branch must execute first to determine which instruction to fetch next
    - Tells hardware if branch is taken or not taken
  - Instructions following a conditional branch are control dependent on the branch instruction
    - Usually program executes different instructions if branch is taken or not

Example (quicksort/MIPS)

```
#        for (;  (j < high)  &&  (array[j] < array[low])  ;  ++j  );
#        $10  =  j
#        $9  =  high
#        $6  =  array
#        $8  =  low
         bge        done,  $10,    $9
         mul        $15,   $10,    4
         addu       $24,   $6,     $15
         lw         $25,   0($24)
         mul        $13,   $8,     4
         addu       $14,   $6,     $13
         lw         $15,   0($14)
         bge        done,  $25,    $15
cont:
         addu       $10,  $10,  1
         . . .
done:
         addu       $11,  $11,  -1
```



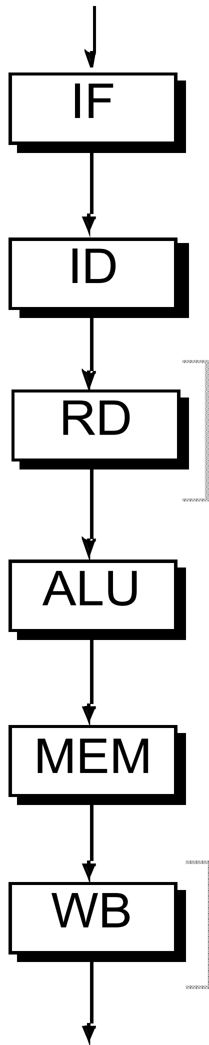
Pipeline Hazards

- Pipeline hazards
  - Potential violations of program dependences
  - Must ensure program dependences are not violated
- Hazard resolution
  - Static: compiler/programmer guarantees correctness
  - Dynamic: hardware performs checks at runtime
- Pipeline interlock
  - Hardware mechanism for dynamic hazard resolution
  - Must detect and enforce dependences at runtime
    - E.g., stall until hazard condition is gone

Pipeline Hazards

- Necessary conditions:
  - WAR: write stage earlier than read stage
    - Is this possible in IF-RD-EX-MEM-WB ?
  - WAW: write stage earlier than write stage
    - Is this possible in IF-RD-EX-MEM-WB ?
  - RAW: read stage earlier than write stage
    - Is this possible in IF-RD-EX-MEM-WB?
- If conditions not met, no need to resolve
- Check for both register and memory

Pipeline Hazard Analysis



WAR: write stage earlier than read?
WAW: write stage earlier than write?
RAW: read stage earlier than write?

- Memory hazards
  - WAR: Yes/No?
  - WAW: Yes/No?
  - RAW: Yes/No?
- Register hazards
  - WAR: Yes/No?
  - WAW: Yes/No?
  - RAW: Yes/No?

RAW Hazard

- Earlier instruction produces a value used by a later instruction:
  - add $1, $2, $3
  - sub $4, $5, $1

Cycle: Instr:	1	2	3	4	5	6	7	8	9	10	11	12	13
add	F	D	X	M	W								
sub		F	D	X	M	W							

RAW Hazard - Stall

- Detect dependence and stall:
  - add $1, $2, $3
  - sub $4, $5, $1

Cycle: Instr:	1	2	3	4	5	6	7	8	9	1 0	1 1	1 2	1 3
add	F	D	X	M	W								
sub						F	D	X	M	W			

Control Dependence

- One instruction affects which executes next
  - sw $4, 0($5)
  - bne $2, $3, loop
  - sub $6, $7, $8

Cycle: Instr:	1	2	3	4	5	6	7	8	9	10	11	12	13
sw	F	D	X	M	W								
bne		F	D	X	M	W							
sub			F	D	X	M	W						

Control Dependence - Stall

- Detect dependence and stall
  - sw $4, 0($5)
  - bne $2, $3, loop
  - sub $6, $7, $8

Cycle: Instr:	1	2	3	4	5	6	7	8	9	10	11	12	13
sw	F	D	X	M	W								
bne		F	D	X	M	W							
sub					F	D	X	M	W				

CS/ECE 552: Pipeline Hazards Part 2

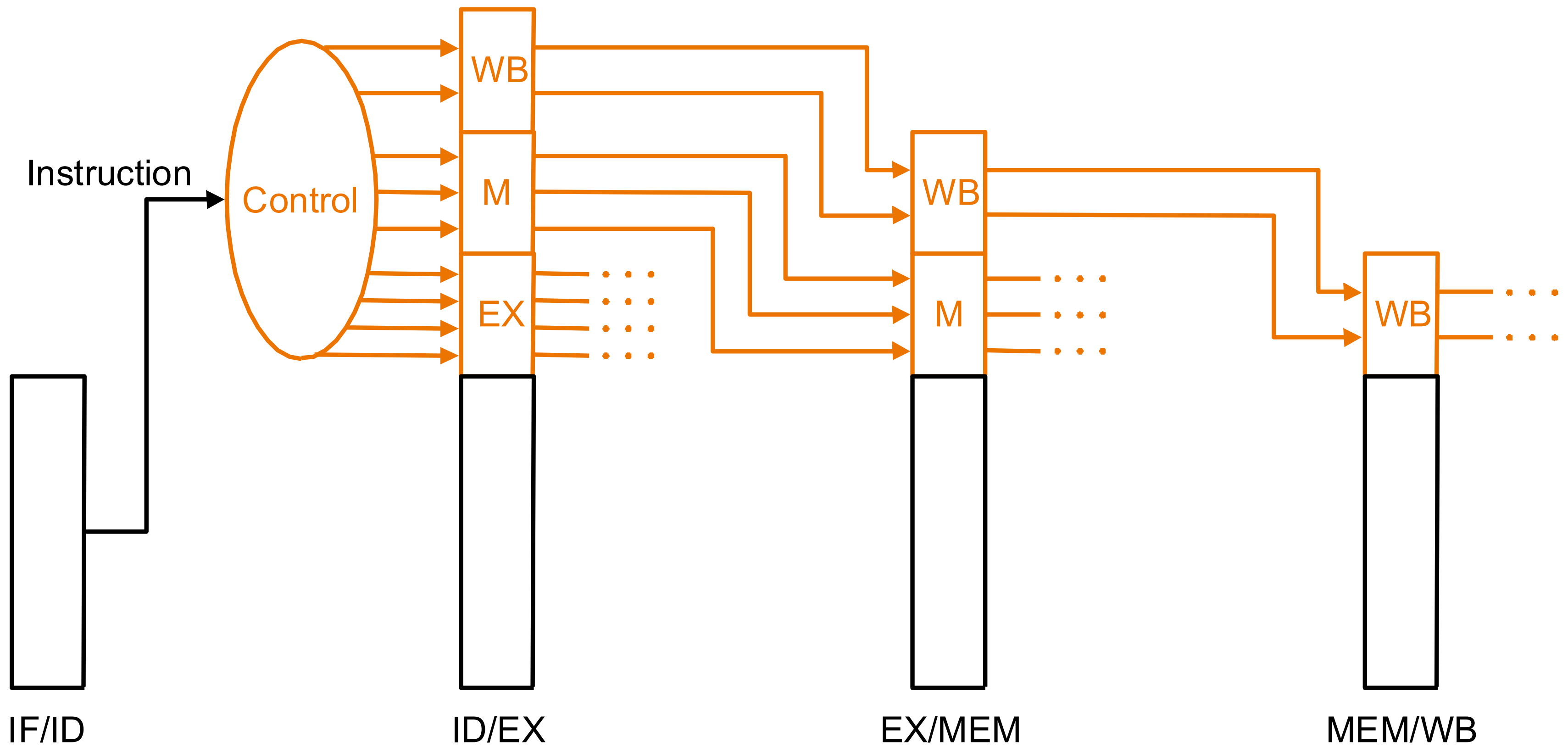
Prof. Matthew D. Sinclair

Lecture notes based in part on slides created by Mark Hill,
Mikko Lipasti, David Wood, Guri Sohi,
John Shen and Jim Smith

Pipelined Control

- Each stage controlled by a different instruction
- Decode instruction in ID, pass its control signals through pipeline
- Control sequencing embedded in pipeline
  - No explicit FSM
  - Instead, distributed FSM (harder to verify)

Pipelined Control



RAW Hazards

- Must first detect RAW hazards
  - Pipeline analysis proved that WAR/WAW don't occur

ID/EX.WriteRegister = IF/ID.ReadRegister1

ID/EX.WriteRegister = IF/ID.ReadRegister2

EX/MEM.WriteRegister = IF/ID.ReadRegister1

EX/MEM.WriteRegister = IF/ID.ReadRegister2

MEM/WB.WriteRegister = IF/ID.ReadRegister1

MEM/WB.WriteRegister = IF/ID.ReadRegister2

RAW Hazards

- Not all hazards because
  - WriteRegister not used (e.g., sw, branch)
  - ReadRegister not used (e.g., addi, jump)
  - Do something only if necessary
    - Logic becomes more complicated than previous slide

RAW Hazards

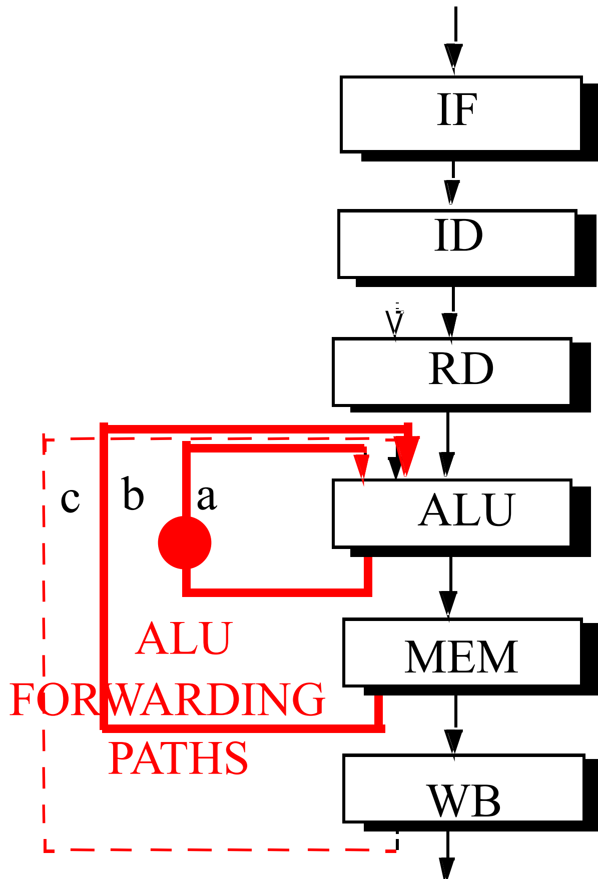
- Hazard Detection Unit
  - Several 5-bit comparators
- Response? Stall pipeline
  - Instructions in IF and ID stay
  - IF/ID pipeline register not updated
  - Send ‘nop’ down pipeline (called a bubble)
  - HW Changes: PCWrite, IF/ID.Write, and nop mux
  - X, M, WB instructions continue

RAW Hazard Forwarding

- A better response – forwarding
  - Also called bypassing
- Stalling: relies on comparators to ensure register is read after it is written
- Forwarding: don't stall, forward!
  - Use mux to select forwarded value from later pipeline stage instead of register value
  - Control mux with hazard detection logic

Forwarding Paths (ALU instructions)

THE UNIVERSITY of WISCONSIN MADISON

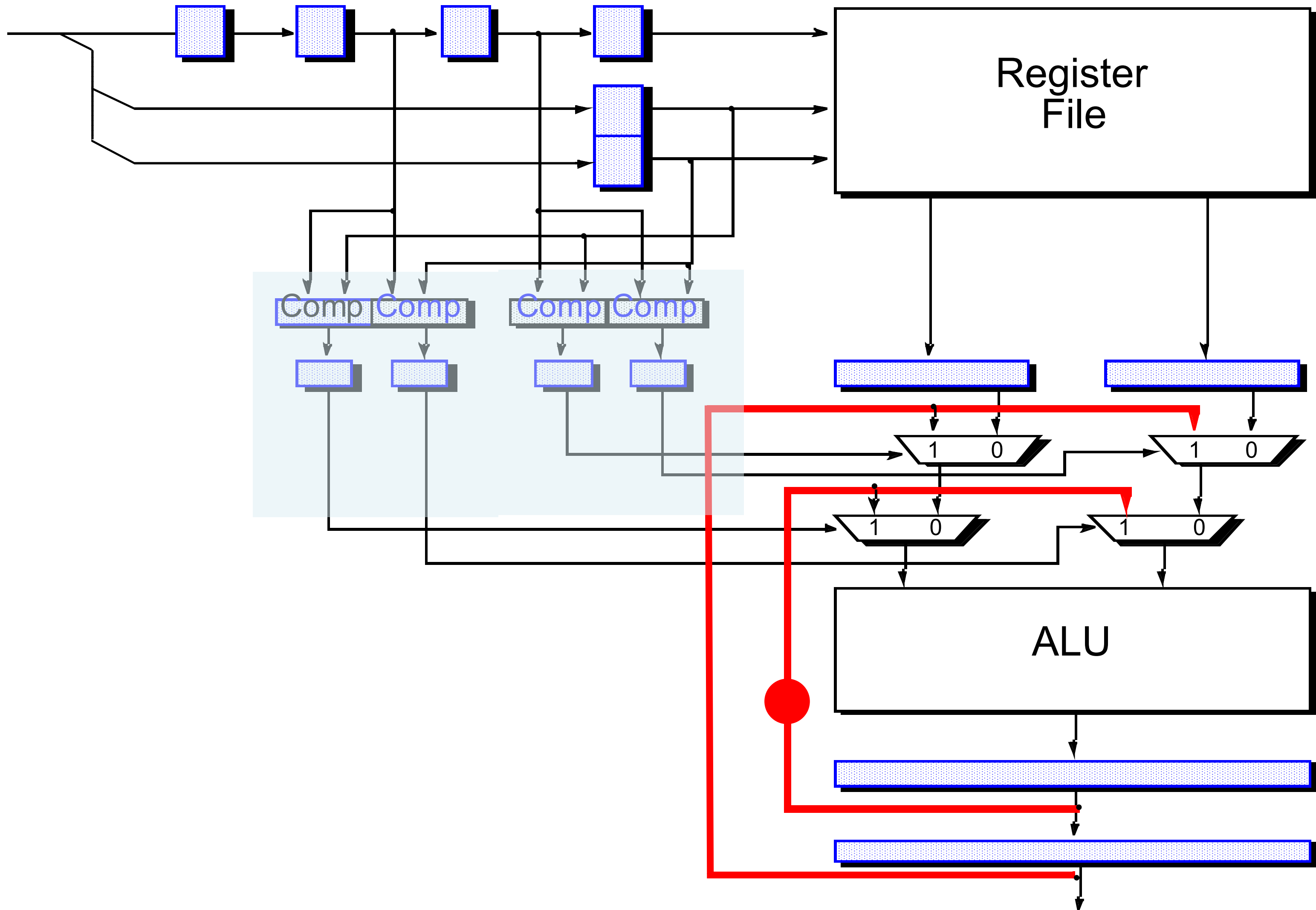


i+1: ← R1	i+2: ← R1	i+3: ← R1
i: R1 ←	i+1:	i+2: ← R1
	i: R1 ←	i+1: ←
		i: R1 ←
(i → i+1)	**(i → i+2)**	**(i → i+3)**
Forwarding via Path a	**Forwarding via Path b**	**i writes R1 before i+3 reads R1**

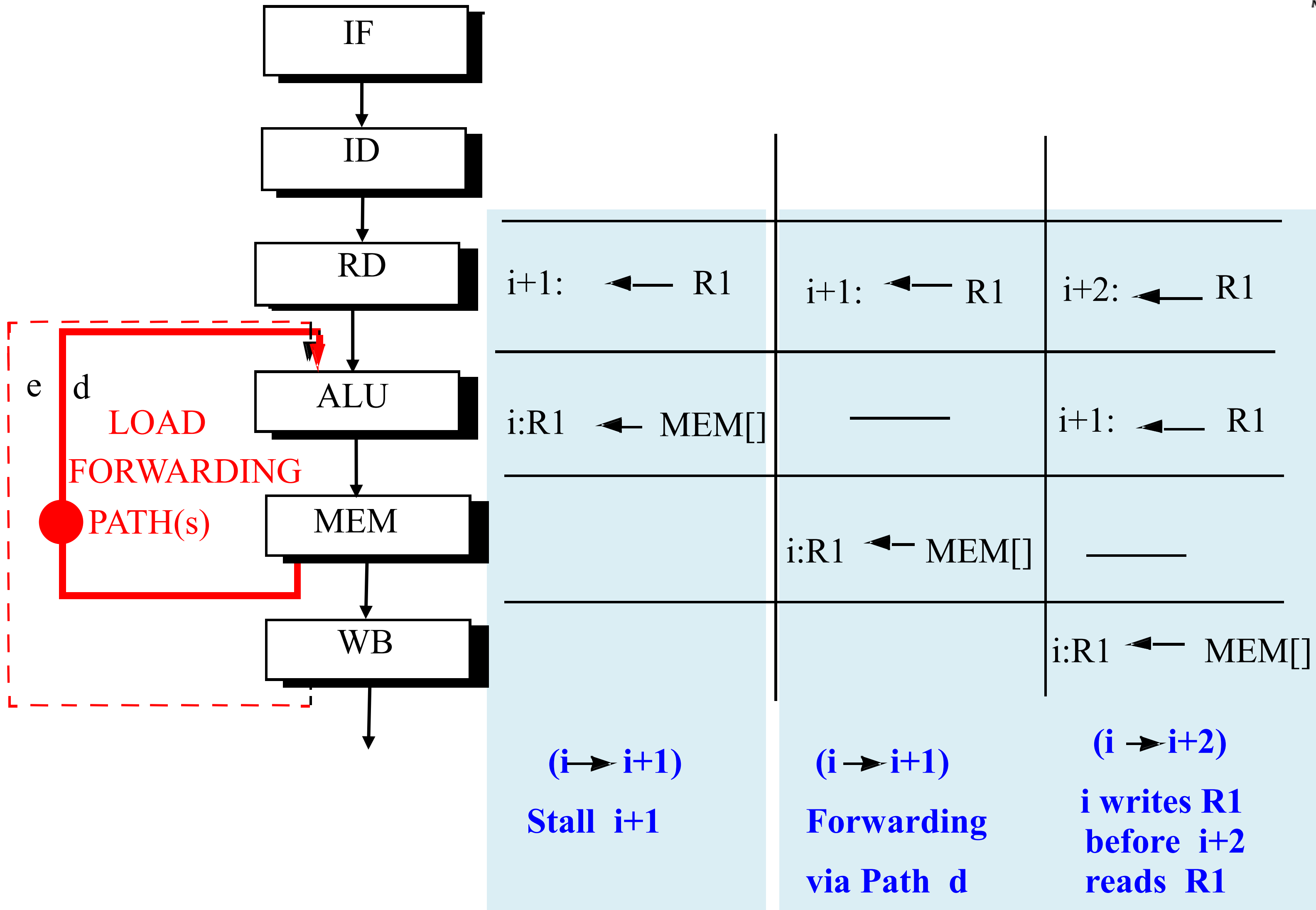
Write before Read RF

- Register file design
  - 2-phase clocks common
  - Write RF on first phase
  - Read RF on second phase
- Hence, same cycle:
  - Write $1
  - Read $1
- No bypass needed
  - If read before write or DFF-based, need bypass

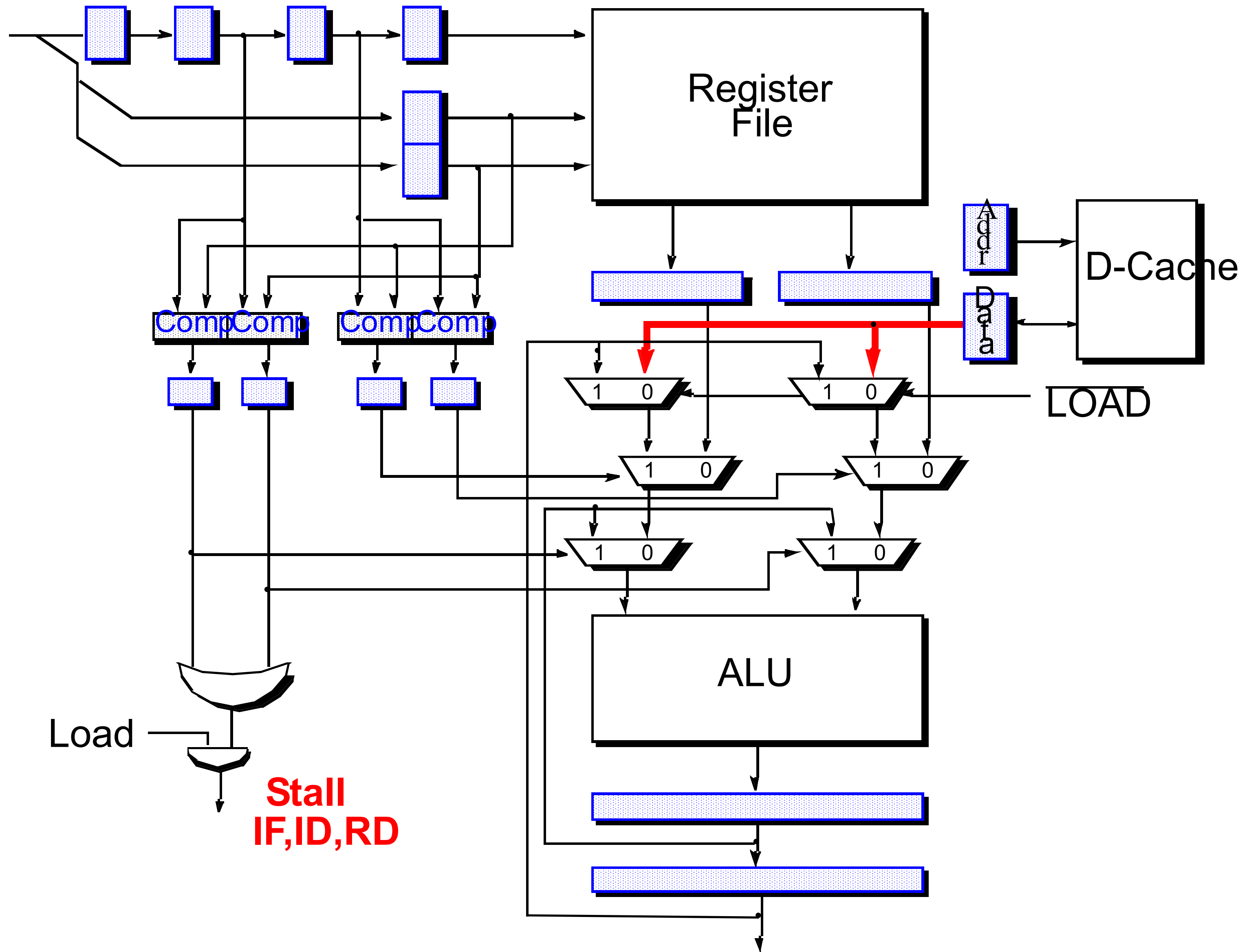
ALU Forwarding



Forwarding Paths (Load instructions)



Implementation of Load Forwarding



CS/ECE 552: Pipeline Hazards Part 3

Prof. Matthew D. Sinclair

Lecture notes based in part on slides created by Mark Hill, Mikko Lipasti, David Wood, Guri Sohi, John Shen and Jim Smith

Control Flow Hazards

- Control flow instructions
  - branches, jumps, jals, returns
  - Can’t fetch until branch outcome is known
  - Too late for next IF

Control Flow Hazards (Cont.)

- What to do?
  - Always stall
  - Easy to implement
  - Performs poorly
  - 1/6th instructions are branches
    - each branch takes 3 cycles
  - CPI = 1 + 3 x 1/6 = 1.5 (lower bound)

Control Flow Hazards (Cont.)

- Predict branch not taken
- Send sequential instructions down pipeline
- Kill instructions later if incorrect
- Must stop memory accesses and RF writes
- Late flush of instructions on misprediction
  - Complex
  - Global signal (wire delay)

Control Flow Hazards (Cont.)

- Even better but more complex
  - Predict taken
  - Predict both (eager execution)
  - Predict one or the other dynamically
    - Adapt to program branch patterns
    - Lots of chip real estate these days
      - Core i7, ARM A15 and their successors
    - Current research topic
  - More later, covered in detail in CS/ECE 752

Control Flow Hazards (Cont.)

- Another option: delayed branches
  - Always execute following instruction
  - “delay slot” (later example on MIPS pipeline)
  - Put useful instruction there, otherwise ‘nop’
- A mistake to cement this into ISA
  - Just a stopgap (one cycle, one instruction)
  - Superscalar processors (later)
    - Delay slot just gets in the way

Exceptions and Pipelining

- add \$1, \$2, \$3 overflows
- A surprise branch
  - Earlier instructions flow to completion
  - Kill (flush) later instructions
  - Save PC in EPC, set PC to Exception handler, etc.
- Costs a lot of designer sanity

Exceptions

- Even worse: in one cycle
  - I/O interrupt
  - User trap to OS (EX)
  - Illegal instruction (ID)
  - Arithmetic overflow
  - Hardware error
  - Etc.
- Interrupt priorities must be supported

Pipeline Hazards

THE UNIVERSITY of WISCONSIN MADISON

- Program Dependences
- Data Hazards
  - Stalls
  - Forwarding
- Control Hazards
  - Stalls
  - Speculation
- Exceptions