CS305
Computer Architecture

Pipeline Control

Bhaskaran Raman
Room 406, KR Building
Department of CSE, IIT Bombay

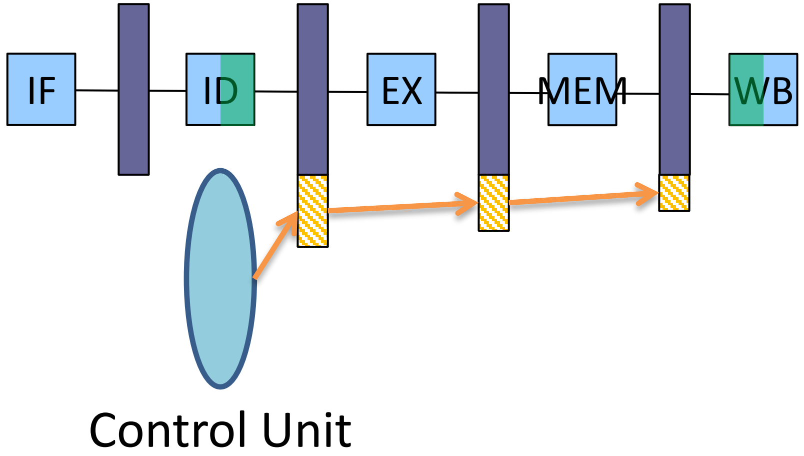
http://www.cse.iitb.ac.in/~br

Controlling the Pipeline

- Without hazards: simple ☺
- Principle: carry your work with you!

Carry Your Work With You

Single cycle control lines:
ALUSrc, ALUIP4: for EX stage branch
MemRd, MemWr: for MEM stage
RegWr, RegDst, Mem2Reg: for WB stage



Pipeline Control to Handle Hazards

- Single-cycle control: combinational circuit
- Multi-cycle control: state machine
- Pipeline control: micro-code (Turing machine)
- Recall: micro-code is code executed by a small/simple processor within the main processor (to general control lines for the pipelined main processor)

Pipeline Control for Data Forwarding to EX

ADD R1, R2, R3

SUB R4, R1, R5

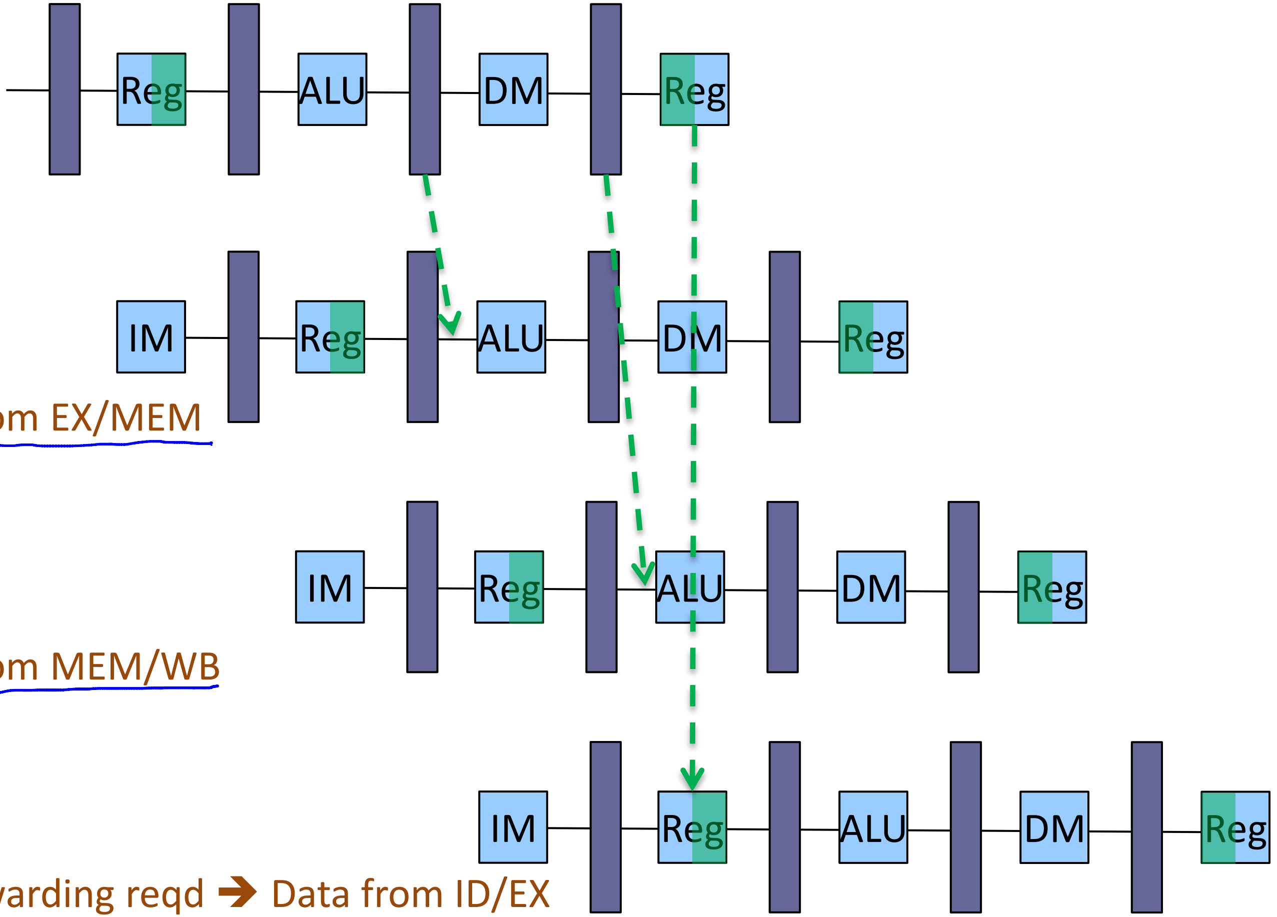
Data from K-1 ➔ Data from EX/MEM

AND R6, R1, R7

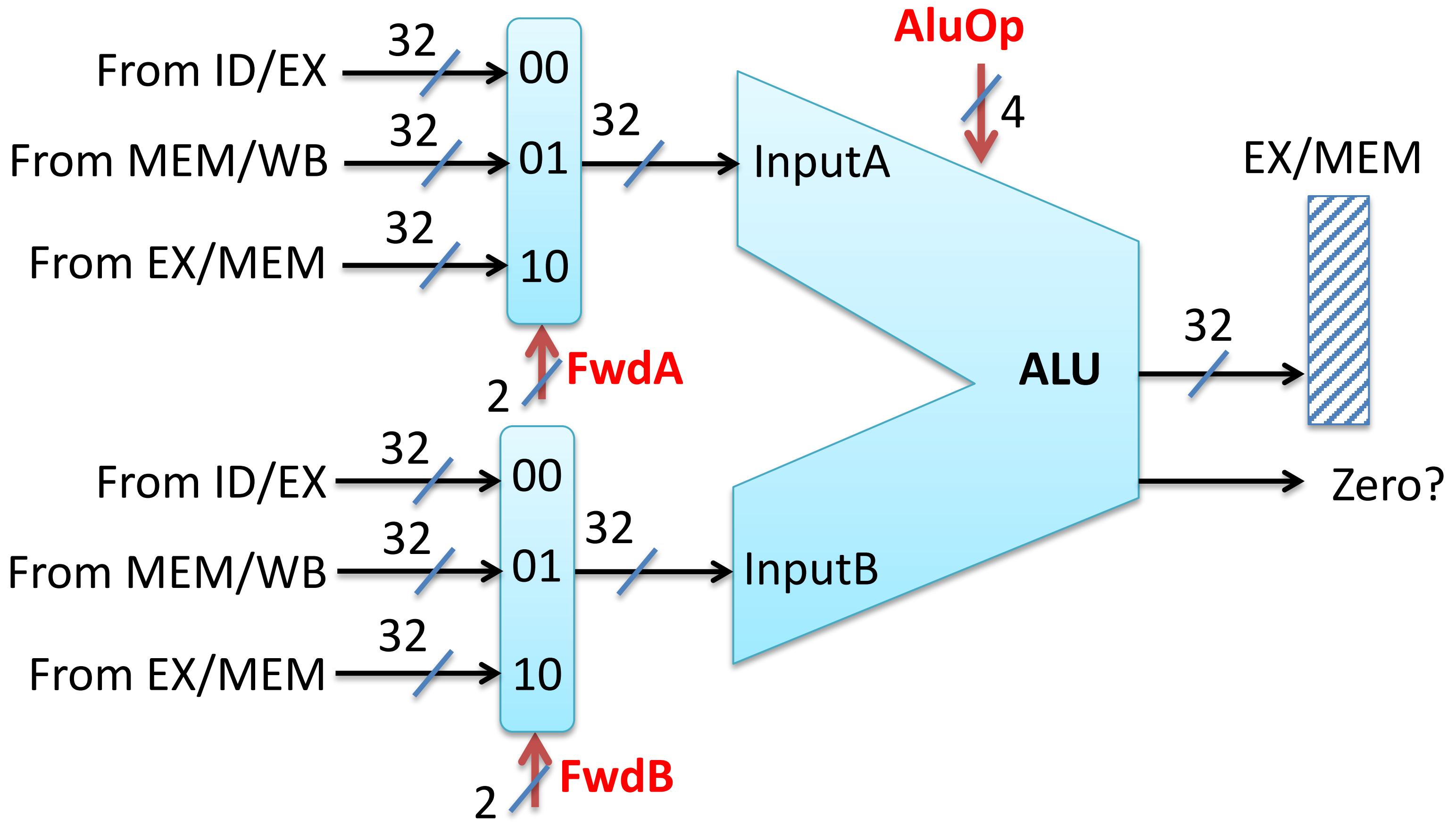
Data from K-2 ➔ Data from MEM/WB

OR R8, R1, R9

Data from K-3 ➔ No forwarding reqd ➔ Data from ID/EX



Forwarding to EX: Data-path Changes



Micro-coded Control for Forwarding to EX

Q: What control lines to generate?

A: FwdA, FwdB

Q: When should these be generated?

A: In ID stage of dependent instrn.

Q: What variables to use?

A: Latches, parts of latches

Q: What is the logic to implement?

A: Cases of no fwd, fwd from K-1, K-2

ADD	R1, R2, R3
SUB	R1, R4, R5
SLT	R6, R1, R0

```
// FwdA: micro-coded control
// Case-1: no forwarding
FwdA = 00;
// Case-2: fwd from K-1
if((IF/ID.Rs==ID/EX.Rd) &&
    (ID/EX.RegWr==1) &&
    (ID/EX.MemRd==0))
      FwdA = 10; //frm EX/MEM
// Case-3: fwd from K-2
if((IF/ID.Rs==EX/MEM.Rd) &&
    (EX/MEM.RegWr==1) &&
    (EX/MEM.MemRd==0))
      FwdA = 01; //frm MEM/WB
```

Swap order

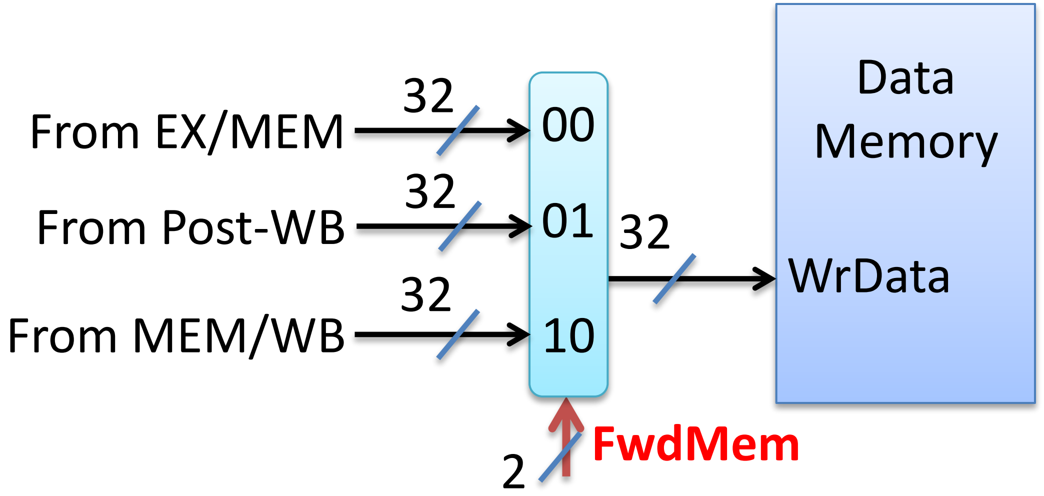
Q: Change for FwdB? **A: Rs → Rt**

Why is Pipeline Control Logic Difficult?

- A lot of subtle possibilities ➔ need attention to detail
- Things to keep in mind:
  - Logic runs in ID stage of dependent instruction
  - Actual forwarding happens later
  - Beware of multiple dependences
  - Beware of assumptions
- Exercise: add support for forwarding to EX from **`lw`**

Forwarding to MEM: Data-path Changes

	CC1	CC2	CC3	CC4	CC5
`OR  R1, R4, R5`	MEM	WB			
`SLL R1, R2, 3`	EX	MEM	WB		
`ADD R1, R2, R3`	ID	EX	MEM	WB	
`SW  R1, 4(R20)`	IF	ID	EX	MEM	WB



Forwarding to MEM: Micro-coded Control

```
// FwdMem: micro-coded control
// Case-1: no forwarding
FwdMem = 00;
// Case-2: fwd from K-2
if((IF/ID.Rt==EX/MEM.Rd) &&      ((CtlUnit.MemWr==1)
    (EX/MEM.RegWr==1) &&          (IF/ID.opcode == SW)
    (EX/MEM.MemRd==0))
      FwdMem = 01; //frm Post-WB
// Case-3: fwd from K-1
if((IF/ID.Rt==ID/EX.Rd) &&       ((CtlUnit.MemWr ==1)
    (ID/EX.RegWr==1) &&
    (ID/EX.MemRd==0))
      FwdMem = 10; //frm MEM/WB
```

So Far…

- Pipeline control without hazards
- Pipeline control for data forwarding
  - Pseduo-micro-code
- Next: pipeline control for stalling

Stalling Logic: Dependent `reg-reg` After `lw`

```
// Stalling: micro-coded control
// Case-1: dependent Rs
if((IF/ID.Rs==ID/EX.Rt) &&
    (ID/EX.MemRd==1))
      STALL // What does this mean?
// Case-2: dependent Rt
if((IF/ID.Rt==ID/EX.Rt) &&
    (ID/EX.MemRd==1))
      STALL // What does this mean?
```

What does STALL mean?

- Do nothing ➔ write nothing
  - `nop` proceeds in the pipeline
  - Zero out control signals, specifically RegWr, MemWr, MemRd
    - No change to machine state
- IF and ID stages must repeat
  - Disable PCWr
  - Disable writing of IF/ID latch

```
// STALL pseudo-micro-code
PCWr = 0;
IF/ID.Wr = 0;
ID/EX latch = 0; // nop (bubble) in pipeline
```

Stalling Logic for Control Hazard

```
if (CtlUnit.branch == 1)
     IF/ID latch = 0; // 1st nop follows branch
if (ID/EX.branch == 1)
     IF/ID latch = 0; // 2nd nop follows branch
```

Q: Changes for 2-stage branch?

A: Second if condition unnecessary

Q: Diff from data hazard stall?

A: nop FOLLOWS branch

Summary

- Pipeline control: subtle logic, involving many details
  - Data forwarding logic
    - Control lines generated in ID stage, actual forwarding may happen later
  - Stalling logic: **`nop`** before stalling instruction, **`nop`** after branch
- We’ve seen only bits and pieces
- Very difficult to write control logic without micro-code
- Next: exceptions, the bane of all pipelines