E0-243: Computer Architecture

Instructors:

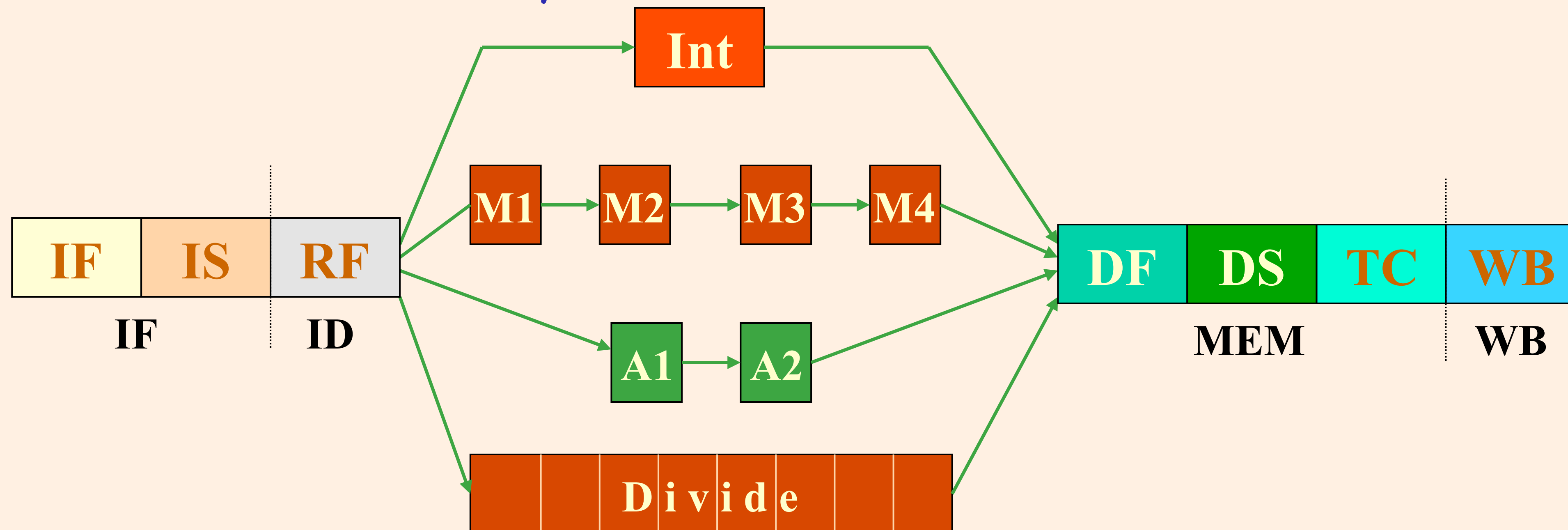
Arkaprava Basu

R. Govindarajan

Instruction-Level Parallelism : Review

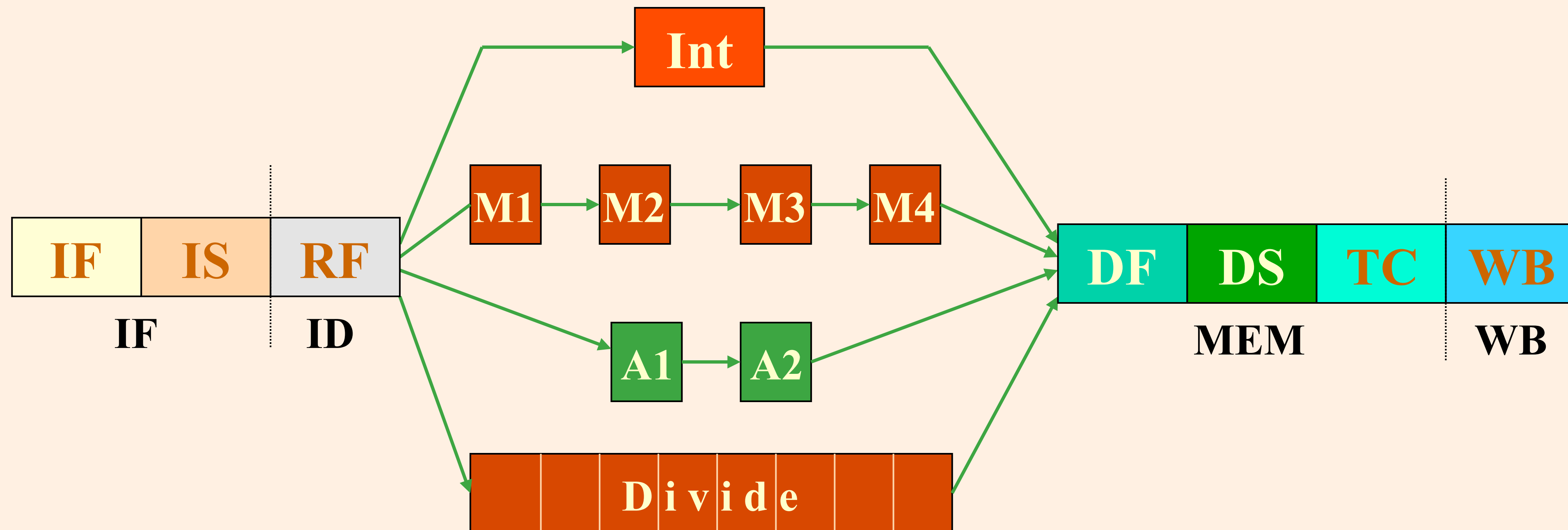
Multi-ALU Organization

- More realistic pipeline with different ALUs
- Possible structural hazards at MEM stage & divide
- Branch Target Addr. calculation and branch condition resolution in EX stage
- Branch stall 3 cycles



Data Hazards

- RAW hazards can result in more than 1 stall cycle, e.g., betn. Divide and Add
- Load and Dependent instructions lead to 3 stall cycles
- WAR and WAW hazards also possible!



Instruction Reordering

Example: Divd F4, F2, F0
Subd F8, F6, F4
Multd F10, F2, F6

MultD could be issued even though *Subd* is stalled.

- Instruction reordering to avoid/reduce stalls
  - Static Instruction Scheduling (compile time)
  - Dynamic Instruction Scheduling (runtime – hardware) : Ability to issue and execute instrns. that follow (in the program order) a stalled instrn.
    - In-order vs. Out-of-order instrn. issue & execution.
      - Scoreboarding (CDC6600), Tomasulo (IBM360)
    - In-order vs. Out-of-order completion

Dynamic Scheduling

- Out-of-order execution divides ID stage:
  - Decode — decode instructions, check for structural hazards.
    - Structural hazard detected by check FU status
  - Issue — wait until data hazards are resolved (checking Reg. Status info); read oprnds.
    - Instrns. wait in a Issue Queue and Read Register Operands after they are ready (or)
    - Instrns. wait in Reservation Stations and operands, as and when they become available, copied in the reservation stations -- avoids WAR and WAW

Dynamic Instrn. Scheduling

Instrn. Buffer

FP Registers

FP Ops.

Store Buffers

Addr. Unit

3
2
1

Load Buffers

1
2

Reservation Stations

Memory

FP adders

FP multipliers

Common Data Bus (CDB)

Dynamic Scheduling

- Execute: Execution begins after read operands.
- Writeback:
  - Communicate results to waiting instrns. (in Reservation Stations or Instrn. Queue) through Common Data Bus
  - Result write also in to destination register
- In-order vs. out-of-order completion/commit

Why In-order Completion?

- With out-of-order execution, what happens if an earlier instrn. raises an exception after a latter instruction has modified the destination?

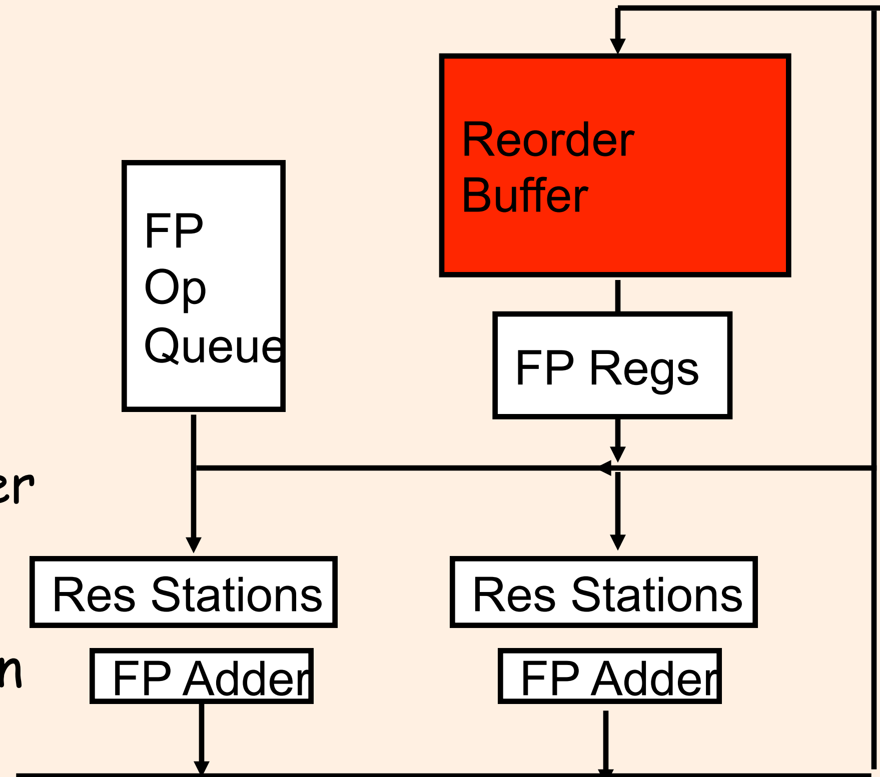
  Example: Divd F4, F2, F0
  Multd F10, F10, F6

  Divd raises an exception after Multd Completes? .

- Precise exception handling
- What happens if speculatively executed instrns. complete (and write result in destination register), before mis-speculation is detected?
  - Architectural state should not be affected by speculative instructions

HW support for In-Order Commit

- Need HW buffer for results of uncommitted instructions: *Reorder Buffer (ROB)*
  - One location reserved in ROB for each dispatched instrn.
  - Write result value in reorder buffer location when execution completes
  - Instrn. Commit in program order instruction commits, result is put into register
  - ROB Supplies operands between execution complete & commit
  - Helps to avoid WAR and WAW
  - Easy to undo speculated instructions on mispredicted branches



Adopted from DAP -CS252-F00@cs.berkeley.edu

Static Scheduling: An Example

- Consider

```
for (i=0; i < n; i++)
    a[i] = a[i] + s;
```

- Assembly code

```
L: LD    F0, 0(R1)          ; 1 stall cycle
   ADDD  F4,F2,F0           ; 3 stall cycle
   ST    0(R1), F4
   ADD   R1, R1, #8
   Sub R2,R2, #1
   Bneqz R2, L              ; 1 branch stall cycle
```

Compiler Techniques to Reduce Stalls

Instruction Scheduling:

```
ld      F0, 0(R1)
                      1 stall
Addd    F4, F2, F0
                      3 stall
St      0(R1), F4
Add     R1, R1, #8
Sub     R2, R2, #1
Bnez    R2, loop      1 stall
```

5 Stalls –

11 cycles/iter

```
ld      F0, 0(R1)
Add     R1, R1, #8
Addd    F4, F2, F0
Sub     R2, R2, #1    2 stall
St      -8(R1), F4
Beqz    R2, loop      1 stall
```

3 Stalls –

9 cycles/iter

Unroll, Rename, and Schedule

```
ld      F0, 0(R1)
ld      F6, -8(R1)
Addd    F4, F2, F0
Addd    F8, F2, F6
Add     R1, R1, #16
Sub     R2, R2, # 2
St      -16(R1), F4
Beqz    R2, loop
St      -8(R1), F8
```

0 Stalls --
4.5 cycles/iter.

- Renaming increases the reordering possibilities.
- Load, FP and branch stalls avoided by instrn. scheduling.
- Unrolling requires more registers.
- Scheduling increase reg. Requirement.

Superscalar Processors

Reading Material for Aug.16-18 Class:

- J.E. Smith and G.S. Sohi. Microarchitecture of Superscalar Processors. Proceedings of the IEEE, 83(12), 1609-1624.
- K.C. Yeager. The MIPS R10000 Superscalar Processor. IEEE MICRO, 28-40, April 1996.