Lecture 2: Exceptions everywhere

Hossam A. H. Fahmy

Cairo University, Faculty of Engineering

Overview

1. Hazards
   - RAW, RAR, WAW, WAR
   - Control hazards
2. Extended pipelines
   - Multi-cycle execution
3. Exceptions
   - Precise exceptions
4. Looking forward

Where are we?

- It is easier to pipeline instructions that have a fixed format:
  - all the instructions are of the same size
    and
  - in all instructions the fields have a fixed size and occupy fixed locations.
- The use of more pipeline stages increases the frequency of operation but it adds
  - timing overheads
    and
  - more hazards.

Data hazards

What are the dependencies that you see in the following code?

I_1: `DIV R3, R1, R2`
I_2: `ADD R5, R3, R2`
I_3: `MUL R1, R2, R6`
I_4: `ADD R5, R1, R5`
I_5: `MUL R4, R2, R6`

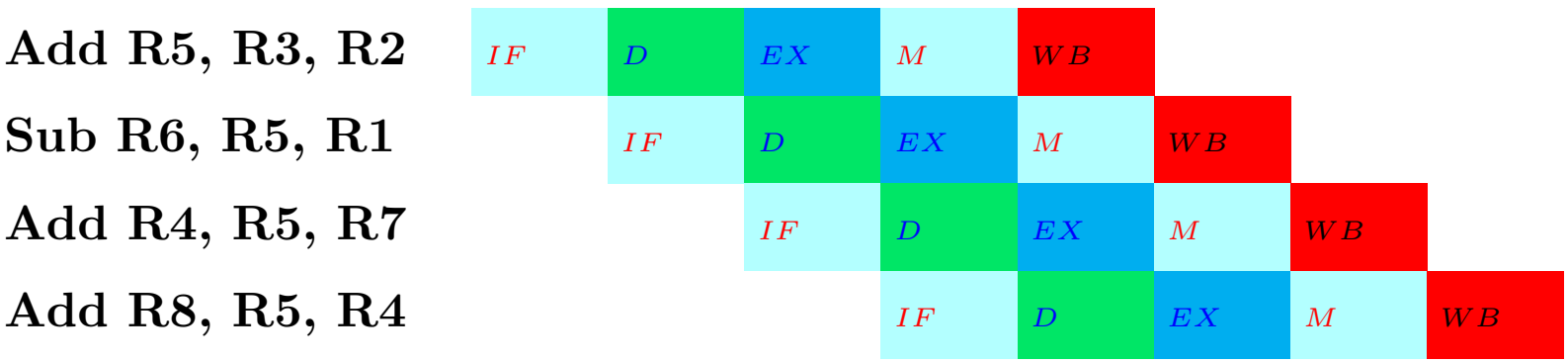
Types of dependencies

If instruction i precedes instruction j and the sources or destinations match then we have a dependency.

	D_i	S_{1_i} or S_{2_i}
S_{1_j} or S_{2_j}	Essential, RAW	RAR
D_j	Output, WAW	Ordering, WAR

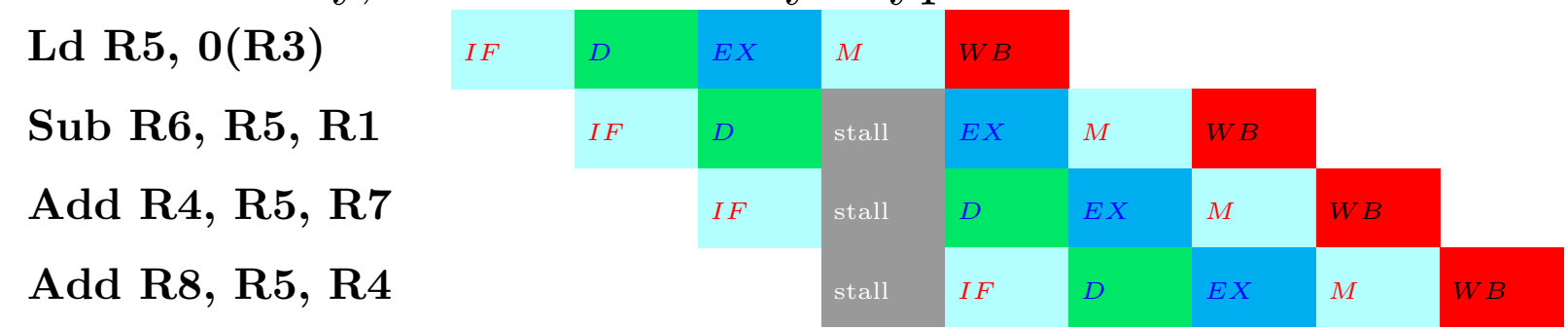
Bypass instead of RAW stalls

It is better to *forward* (or *bypass*) the data instead of stalling.



Some must stall

Unfortunately, we cannot always bypass



Control hazards

- For these we must flush any instructions from the wrong direction
- We will deal with “prediction” in the coming few lectures.

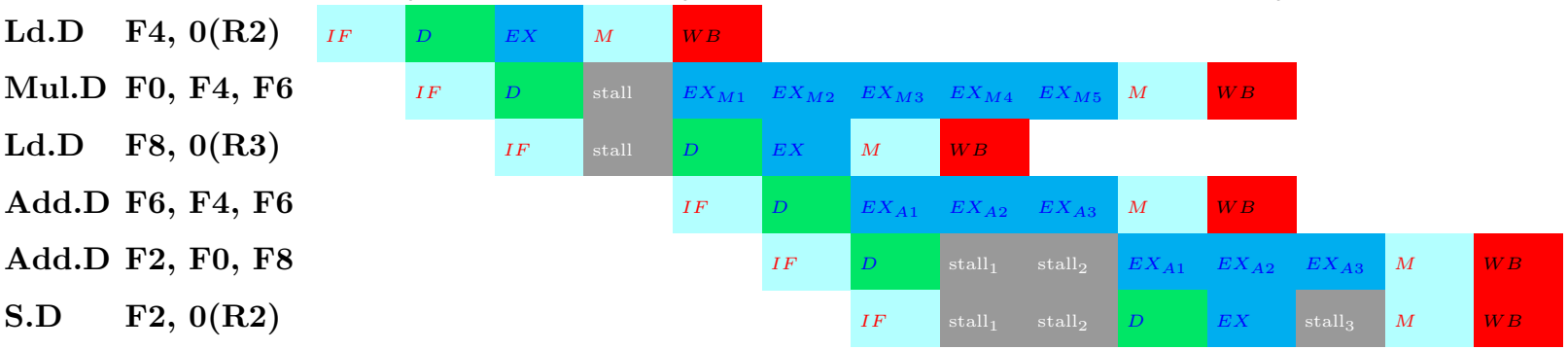
Extending the basic pipeline

We started by forcing all the integer instructions to pass through the same number of stages even if they do not use them. *Why?*
However, the execution of a double precision floating point divide takes from 4 (most aggressive techniques) to over 50 (simple algorithms) cycles.

- Extend the clock cycle. Everything is slow!
- Allow some instructions to take multiple cycles in their execution.

Multi-cycle instructions

Assume multiply takes 5 cycles and add takes 3 cycles.



- A specific unit deals with each of the extended instructions.
- Multi-cycle instructions increase the number of stall cycles.
- Now, we get in order start but out of order termination.
- We may also get multiple instructions in the M or WB stage. $\Rightarrow$ Stall either at the D or at the WB stage.

  Is it really necessary to stall?

The exceptions

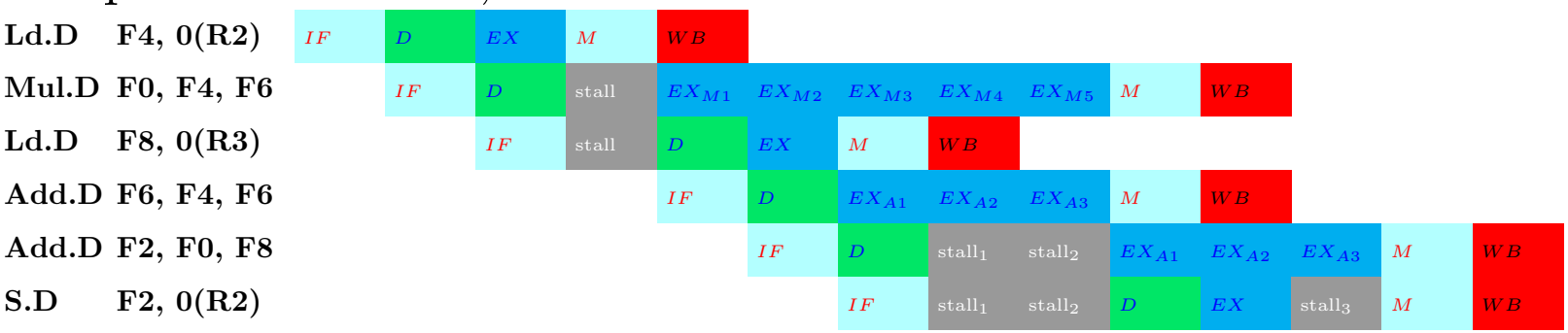
We have external interrupts and internal exceptions. These events have several classifications.

1. User requested versus coerced.
2. Maskable versus nonmaskable.
3. Terminate versus resume.
4. Asynchronous versus synchronous.
5. Between versus within instructions.

In general, the first alternative of these pairs is easier to implement and may be handled after the completion of the current instruction.

Precise exceptions

An exception is precise if all the instructions before the exception finish correctly and all those after it do not change the state. Once the exception is handled, the latter instructions are *restarted* from scratch.



Exception at EX_{A1} of Add.D F6, F4, F6: Allow the Mul.D and Ld.D to complete and *flush* the two Add.D and S.D.

Exception at EX_{M5} of Mul.D: The following Ld.D has already completed! $\Rightarrow$ either force in order WB or "undo".

Both of the above: *Which one has the higher priority? Why?*

Looking forward

- Prediction on the branches.
- Multiple pipelines in parallel.
- Dynamic scheduling of the instructions by the hardware.