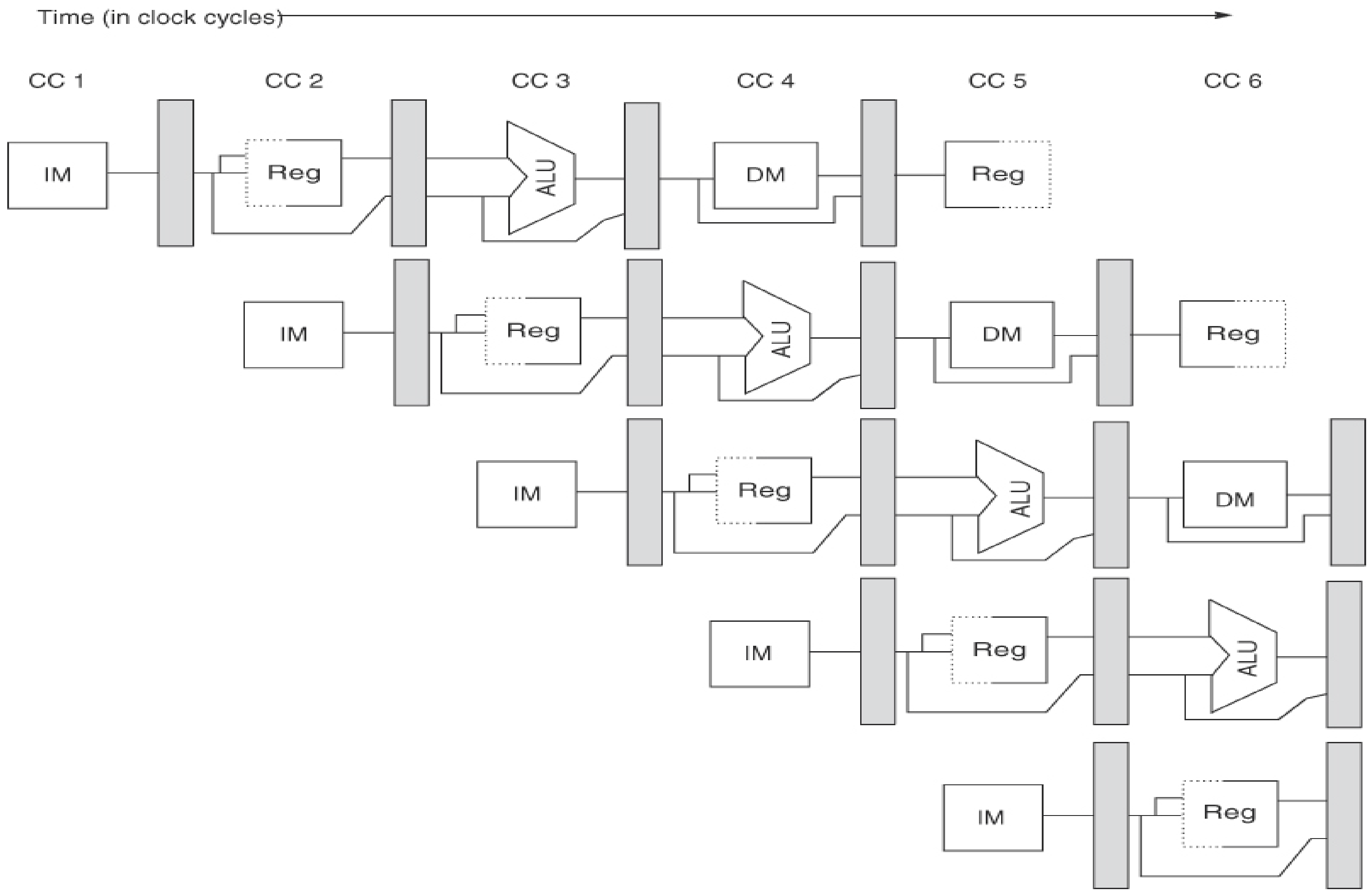
Lecture 18: Pipelining

- Today's topics:
    - 5-stage pipeline
    - Hazards
    - Data dependence handling with bypassing
    - Data dependence examples

A 5-Stage Pipeline



Source: H&P textbook

Performance Improvements?

- Does it take longer to finish each individual job?
- Does it take shorter to finish a series of jobs?
- What assumptions were made while answering these questions?
  - No dependences between instructions
  - Easy to partition circuits into uniform pipeline stages
  - No latch overhead
- Is a 10-stage pipeline better than a 5-stage pipeline?

Quantitative Effects

- As a result of pipelining:
  - Time in ns per instruction goes up
  - Each instruction takes more cycles to execute
  - But… average CPI remains roughly the same
  - Clock speed goes up
  - Total execution time goes down, resulting in lower average time per instruction
  - Under ideal conditions, speedup

    = ratio of *elapsed times between successive instruction completions*

    = number of pipeline stages = increase in clock speed

Hazards

- Structural hazards: different instructions in different stages (or the same stage) conflicting for the same resource
- Data hazards: an instruction cannot continue because it needs a value that has not yet been generated by an earlier instruction
- Control hazard: fetch cannot continue because it does not know the outcome of an earlier branch – special case of a data hazard – separate category because they are treated in different ways

Conflicts/Problems

- I-cache and D-cache are accessed in the same cycle – it helps to implement them separately

- Registers are read and written in the same cycle – easy to deal with if register read/write time equals cycle time/2

- Instructions can’t skip the DM stage, else conflict for RW

- Consuming instruction may have to wait for producer

- Branch target changes only at the end of the second stage -- what do you do in the meantime?

Structural Hazards

- Example: a unified instruction and data cache → stage 4 (MEM) and stage 1 (IF) can never coincide

- The later instruction and all its successors are delayed until a cycle is found when the resource is free → these are pipeline bubbles

- Structural hazards are easy to eliminate – increase the number of resources (for example, implement a separate instruction and data cache, add more register ports)

Data Hazards

- An instruction *produces* a value in a given pipeline stage
- A subsequent instruction *consumes* that value in a pipeline stage
- The consumer may have to be delayed so that the time of consumption is later than the time of production

Example 1 – No Bypassing

- Show the instruction occupying each stage in each cycle (no bypassing) if I1 is R1+R2→R3 and I2 is R3+R4→R5 and I3 is R7+R8→R9

CYC-1	CYC-2	CYC-3	CYC-4	CYC-5	CYC-6	CYC-7	CYC-8
IF	IF	IF	IF	IF	IF	IF	IF
D/R	D/R	D/R	D/R	D/R	D/R	D/R	D/R
ALU	ALU	ALU	ALU	ALU	ALU	ALU	ALU
DM	DM	DM	DM	DM	DM	DM	DM
RW	RW	RW	RW	RW	RW	RW	RW

Example 1 – No Bypassing

- Show the instruction occupying each stage in each cycle (no bypassing) if I1 is R1+R2→R3 and I2 is R3+R4→R5 and I3 is R7+R8→R9

CYC-1	CYC-2	CYC-3	CYC-4	CYC-5	CYC-6	CYC-7	CYC-8
IF I1	IF I2	IF I3	IF I3	IF I3	IF I4	IF I5	IF
D/R	D/R I1	D/R I2	D/R I2	D/R I2	D/R I3	D/R I4	D/R
ALU	ALU	ALU I1	ALU	ALU	ALU I2	ALU I3	ALU
DM	DM	DM	DM I1	DM	DM	DM I2	DM I3
RW	RW	RW	RW	RW I1	RW	RW	RW I2

Example 2 – Bypassing

- Show the instruction occupying each stage in each cycle (with bypassing) if I1 is R1+R2→R3 and I2 is R3+R4→R5 and I3 is R3+R8→R9. Identify the input latch for each input operand.

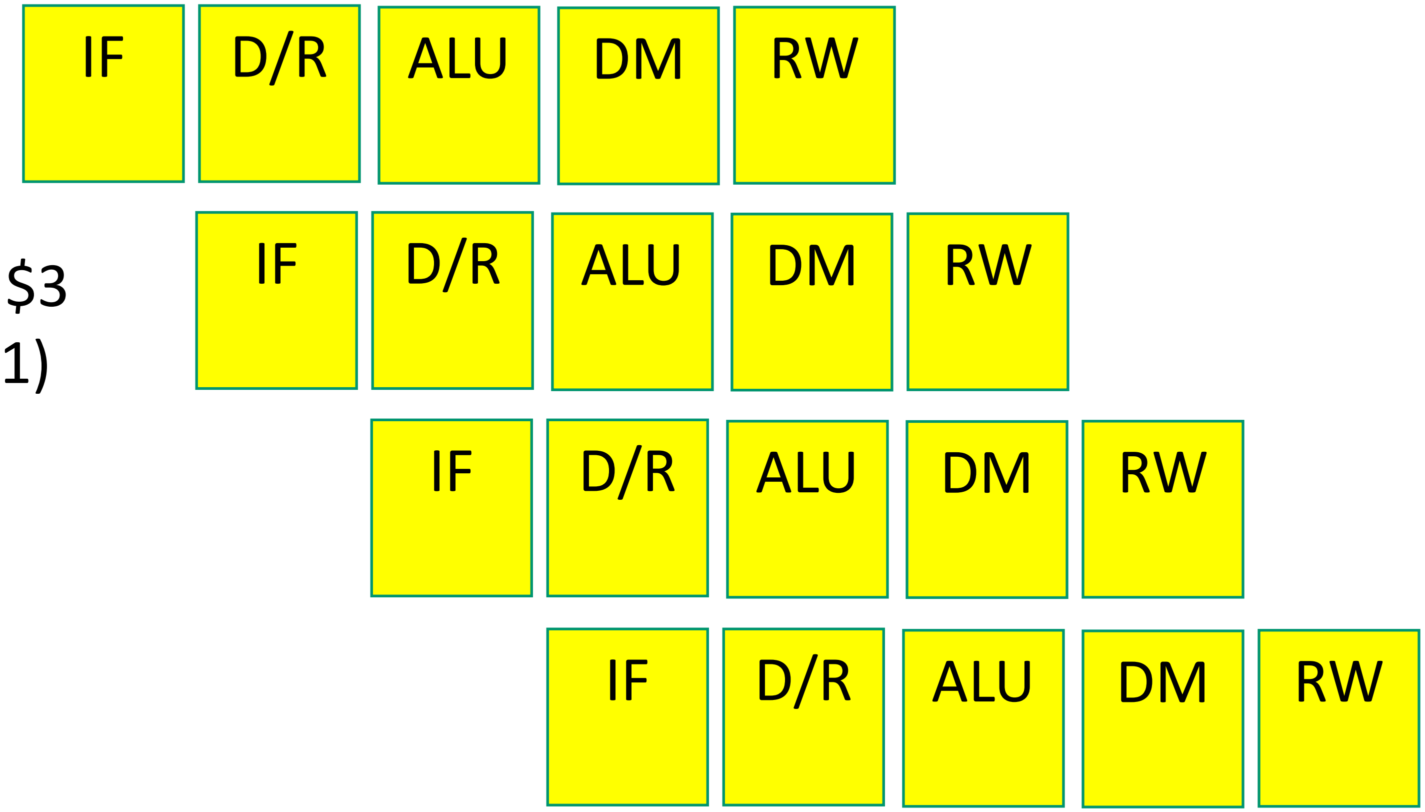
CYC-1	CYC-2	CYC-3	CYC-4	CYC-5	CYC-6	CYC-7	CYC-8
IF	IF	IF	IF	IF	IF	IF	IF
D/R	D/R	D/R	D/R	D/R	D/R	D/R	D/R
ALU	ALU	ALU	ALU	ALU	ALU	ALU	ALU
DM	DM	DM	DM	DM	DM	DM	DM
RW	RW	RW	RW	RW	RW	RW	RW

Example 2 – Bypassing

- Show the instruction occupying each stage in each cycle (with bypassing) if I1 is R1+R2→R3 and I2 is R3+R4→R5 and I3 is R3+R8→R9. Identify the input latch for each input operand.

CYC-1	CYC-2	CYC-3	CYC-4	CYC-5	CYC-6	CYC-7	CYC-8
IF I1	IF I2	IF I3	IF I4	IF I5	IF	IF	IF
D/R	D/R I1	D/R I2	D/R I3	D/R I4	D/R	D/R	D/R
ALU	ALU	**L3 L3** ALU I1	**L4 L3** ALU I2	**L5 L3** ALU I3	ALU	ALU	ALU
DM	DM	DM	DM I1	DM I2	DM I3	DM	DM
RW	RW	RW	RW	RW I1	RW I2	RW I3	RW

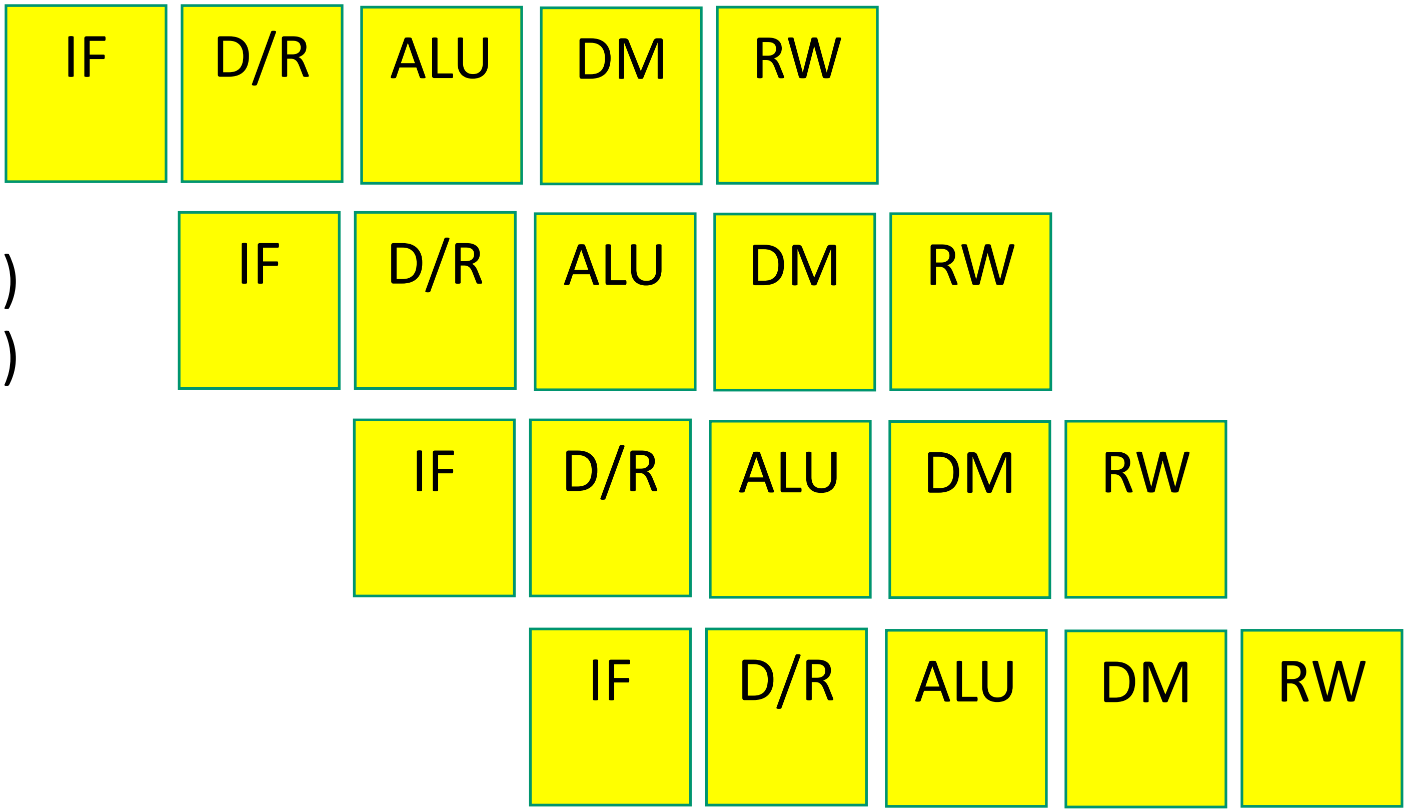
Problem 1



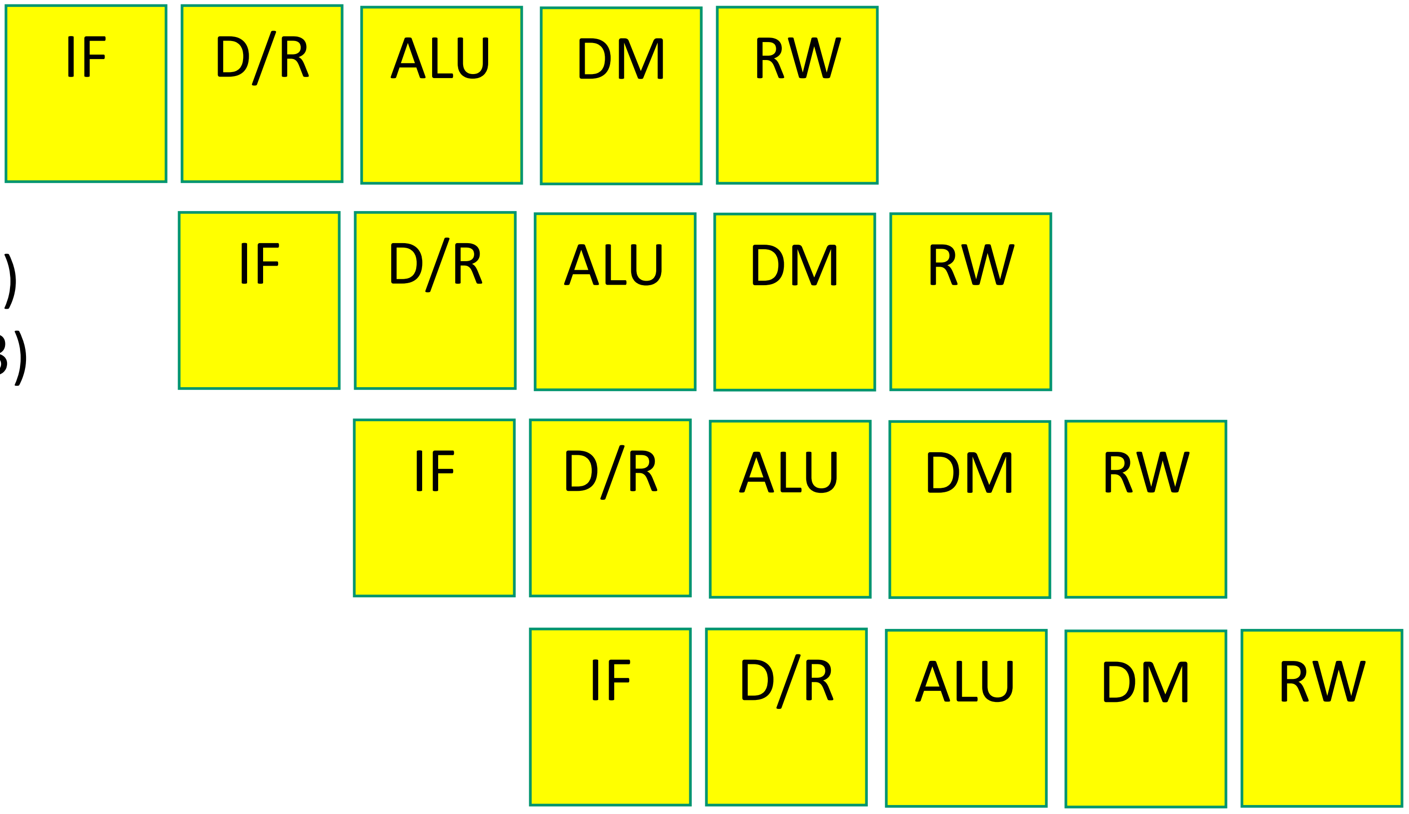
```
add   $1, $2, $3
lw     $4, 8($1)
```

Problem 2

lw $1, 8($2)
lw $4, 8($1)



Problem 3



```
lw   $1, 8($2)
sw   $1, 8($3)
```