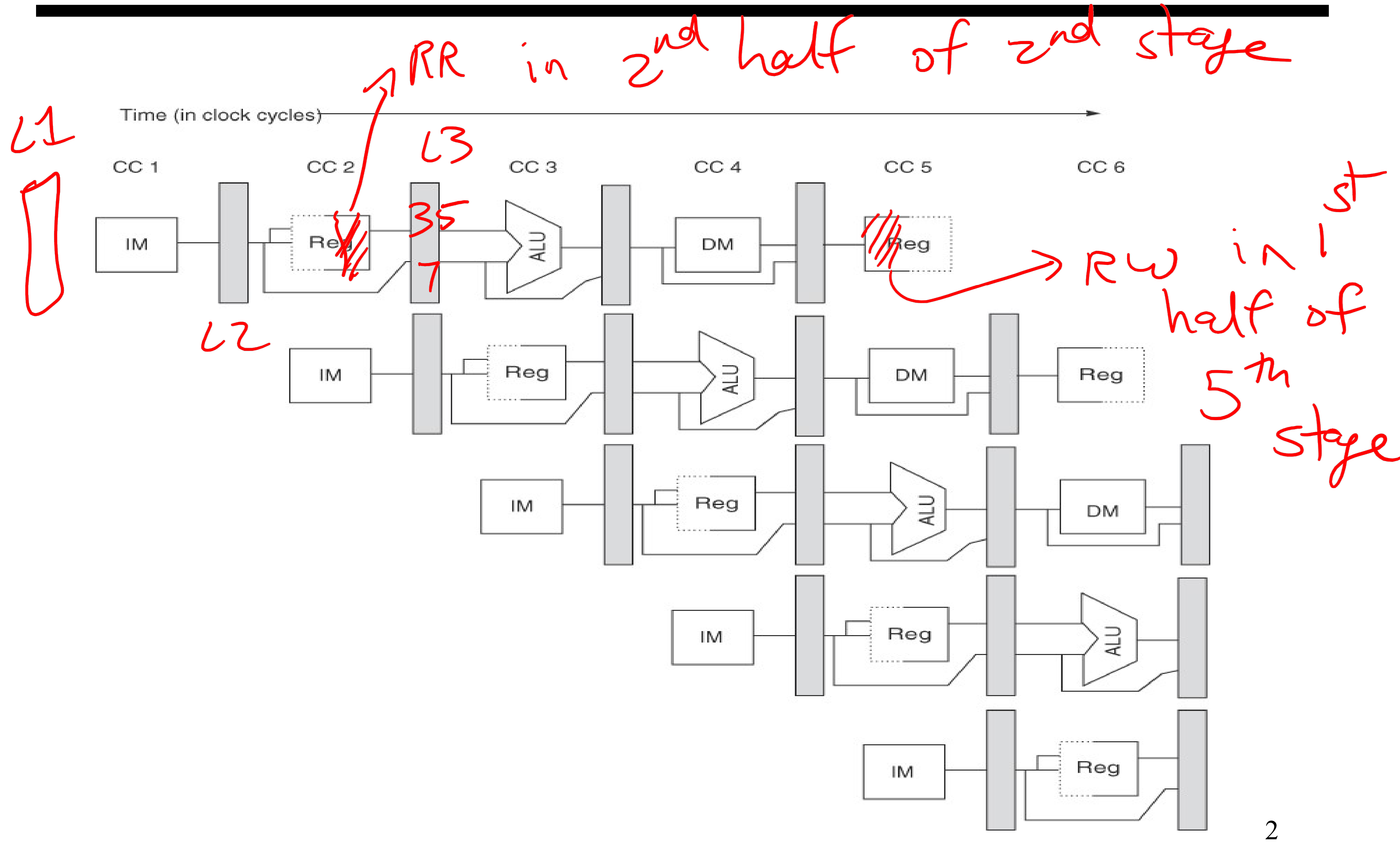
Lecture 17: Pipelining

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

HW7 posted today

Look at "Notes" links on class webpage (FSM, data hazards)

A 5-Stage Pipeline



Source: H&P textbook

Example A

Unpipelined $\rightarrow$ 5-stage pipeline

in theory, 5x speedup

3rd factor: pipelined has more stalls — typical IPC = 0.8

speedup: 3.9, 3.1

- Unpipelined processor: 5ns + 0.2ns latch overhead

  Cycle Time: 5.2ns

  Clock Speed: 1/5.2ns = 180 MHz (assuming ideal conditions) (no stalls)

  CPI: 1 *IPC:* 1

  Throughput: instrs per second

  = IPC × clkspeed = 1 × 180 MHz = 180 million insts / sec

  = 0.18 BIPS

- Pipelined processor: 5 stages, longest stage 1.2ns + 0.2ns latch

  Cycle Time: 1.4 ns

  Clock Speed: 1/1.4ns = 710 MHz (2 factors: latch ovhd, non-uniform stages)

  CPI: 1 *IPC:* 1 (idealized, no-stall assumptions)

  Throughput:

  = IPC × clkspeed

  = 1 × 710 MHz = 710 MIPS = 0.710 BIPS

$$\text{Speedup} = \frac{\text{Thru p}}{\text{Thru u}} = \frac{0.71}{0.18} \approx 3.9x$$

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

Unpipe: 1 instr: 5.2ns
Pipe: 1 instrs: 5 stages × 1.4ns = 7ns

1 cyc → 5 cyc
Unpipe 5.2ns
Pipe 1.4ns

5x 3 factors (latch ovhd, non-unif stages, stall) → 3.1x

Hazards

IM
DM
Reg file
RR
RW
2nd half
1st half

- Structural hazards: different instructions in different stages (or the same stage) conflicting for the same resource

perf vs cost ~~tradeoff~~ tradeoff

Today

- Data hazards: an instruction cannot continue because it needs a value that has not yet been generated by an earlier instruction

add $t1 ←

add ← $t1 + $t2

- 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

Next Week

beq $t1, $t2, offset

Conflicts/Problems

- I-cache and D-cache are accessed in the same cycle – it helps to implement them separately ✓ str haz
- 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 Data haz
- Branch target changes only at the end of the second stage -- what do you do in the meantime? Control haz

Structural Hazards

- Example: a unified instruction and data cache $\rightarrow$ 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 $\rightarrow$ 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

prod in \$t1

producer add \$t1, \$s1, \$s2

- An instruction *produces* a value in a given pipeline stage

consumer add \$t2, \$t1, \$t3

- A subsequent instruction *consumes* that value in a pipeline stage

\$t1 consumed

- The consumer may have to be delayed so that the time of consumption is later than the time of production

Example 1 – No Bypassing

data dep – hazard → stalls $\overset{2}{}$

space

- 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

time ↓

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	IF	IF
D/R Decode	D/R I1 35,7	D/R I2 stall	D/R I2 stall	D/R I2	D/R I3	D/R	D/R
ALU	ALU	ALU I1 42	ALU	ALU BUBBLE 1	ALU I2 BUBBLE 2	ALU I3	ALU
DM	DM	DM	DM I1 42	DM	DM	DM I2	DM I3
RW	RW	RW	RW	RW I1	RW	RW	RW I2

Ignore warmup

3 instr / 5 cyc

$\boxed{= 0.6 \text{ IPC}}$

I3

Example 1 – No Bypassing

HW { R1 + R2 → R3
R7 + R8 → R9
R3 + R4 → R5

IPC = 0.75

compiler sched of insts

- 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

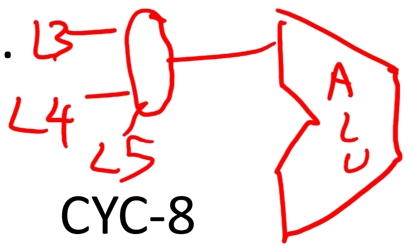
CYC-1	CYC-2	CYC-3	CYC-4	CYC-5	CYC-6	CYC-7	CYC-8
IF I1	IF I2	IF I3	IF I3	IF I3	IF I4	IF I5	IF
D/R	D/R I1	D/R I2	D/R I2	D/R I2	D/R I3	D/R I4	D/R
ALU	ALU	ALU I1	ALU	ALU	ALU I2	ALU I3	ALU
DM	DM	DM	DM I1	DM	DM	DM I2	DM I3
RW	RW	RW	RW	RW I1	RW	RW	RW I2

Forwarding

Example 2 – $\boxed{\text{Bypassing}}$

HW technique

- 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.



(PC)	CYC-1	CYC-2	CYC-3	CYC-4	CYC-5	CYC-6	CYC-7	CYC-8
L1	IF I1	IF I2	IF I3	IF	IF	IF	IF	IF
L2	D/R	D/R I1 35, 7	D/R I2	D/R I3	D/R	D/R	D/R	D/R
L3	ALU	ALU	ALU I1 42	ALU I2	ALU I3	ALU	ALU	ALU
L4	DM	DM	DM	DM I1 42	DM I2	DM	DM	DM
L5	RW	RW	RW	RW	RW R3 I1	RW I2	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.

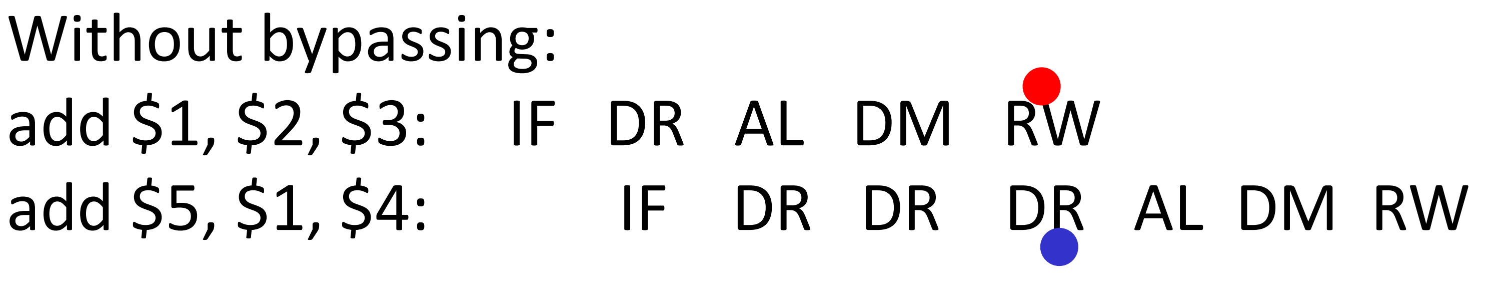
CYC-1	CYC-2	CYC-3	CYC-4	CYC-5	CYC-6	CYC-7	CYC-8
IF I1	IF I2	IF I3	IF I4	IF I5	IF	IF	IF
D/R	D/R I1	D/R I2	D/R I3	D/R I4	D/R	D/R	D/R
		L3 L3	**L4 L3**	**L5 L3**			
ALU	ALU	ALU I1	ALU I2	ALU I3	ALU	ALU	ALU
DM	DM	DM	DM I1	DM I2	DM I3	DM	DM
RW	RW	RW	RW	RW I1	RW I2	RW I3	RW

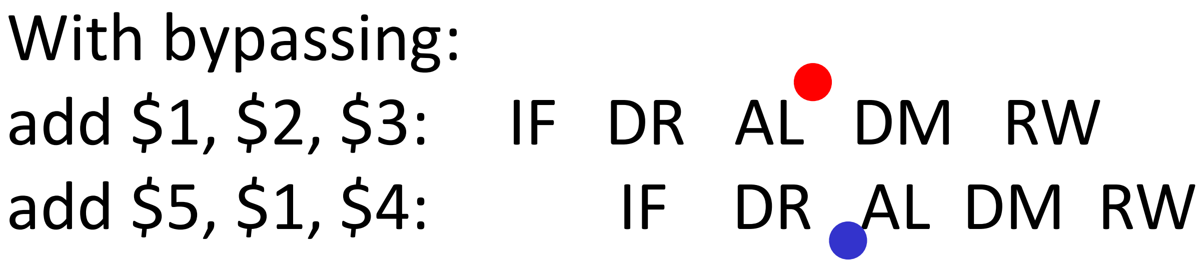
Problem 0

add $1, $2, $3
add $5, $1, $4

● Point of Production

● Point of Consumption


Without bypassing:
add $1, $2, $3: IF DR AL DM RW
add $5, $1, $4: IF DR DR DR AL DM RW


With bypassing:
add $1, $2, $3: IF DR AL DM RW
add $5, $1, $4: IF DR AL DM RW