Lecture 18: Pipelining

Midterms on the table

Last names: A-D E-L M-R S-Z
(for the most part)

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

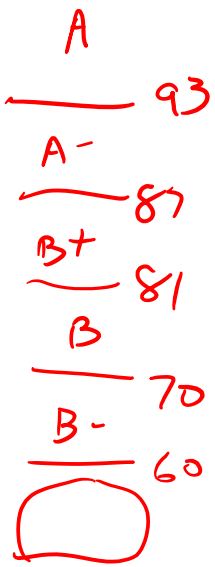
Email me if your midterm score on Canvas is zero!!

HW7 posted later today

Midterm Notes

240 43 45

- Grade categories:
  - Top 18%: 93 (A)
  - Top 36%: 87 (A-)
  - Top 54%: 81 (B+)
  - Top 72%: 70 (B)
  - Top 88%: 60 (B-)



- Common mistakes:
  - Support for subtraction: not including the carry
  - IEEE 754 floating point formats and addition
  - Addition of signed numbers
  - Logic gates, Sum-of-products
  - MARS variables, syscalls
  - Power and energy !!!

$a0
$v0
syscall

Power and Energy

Total Power = Dyn Power + Lkg power

Lkg power $\propto$ Voltage $\rightarrow$ Lkg stays constant at 20W (in midterm question)

Dyn power $\propto$ Act $\times$ C $\times$ f $\times V^2$

100W = 80W dyn + 20W lkg $\Rightarrow$ 2GHz

140W = 80W $\times \frac{3}{2}$ + 20W $\Rightarrow$ 3 GHz Turbo

60W = 80 $\times \frac{1}{2}$ + 20W $\Rightarrow$ 1 GHz

Power $\Rightarrow$ Watts

Prgm takes 10 secs $\Rightarrow$ CPU-bound

At 3GHz, new exec time = $10 \times \frac{2}{3} = 6.66$ secs

At 2GHz, new exec time = $10 \times \frac{2}{1} = 20$ secs

P x t

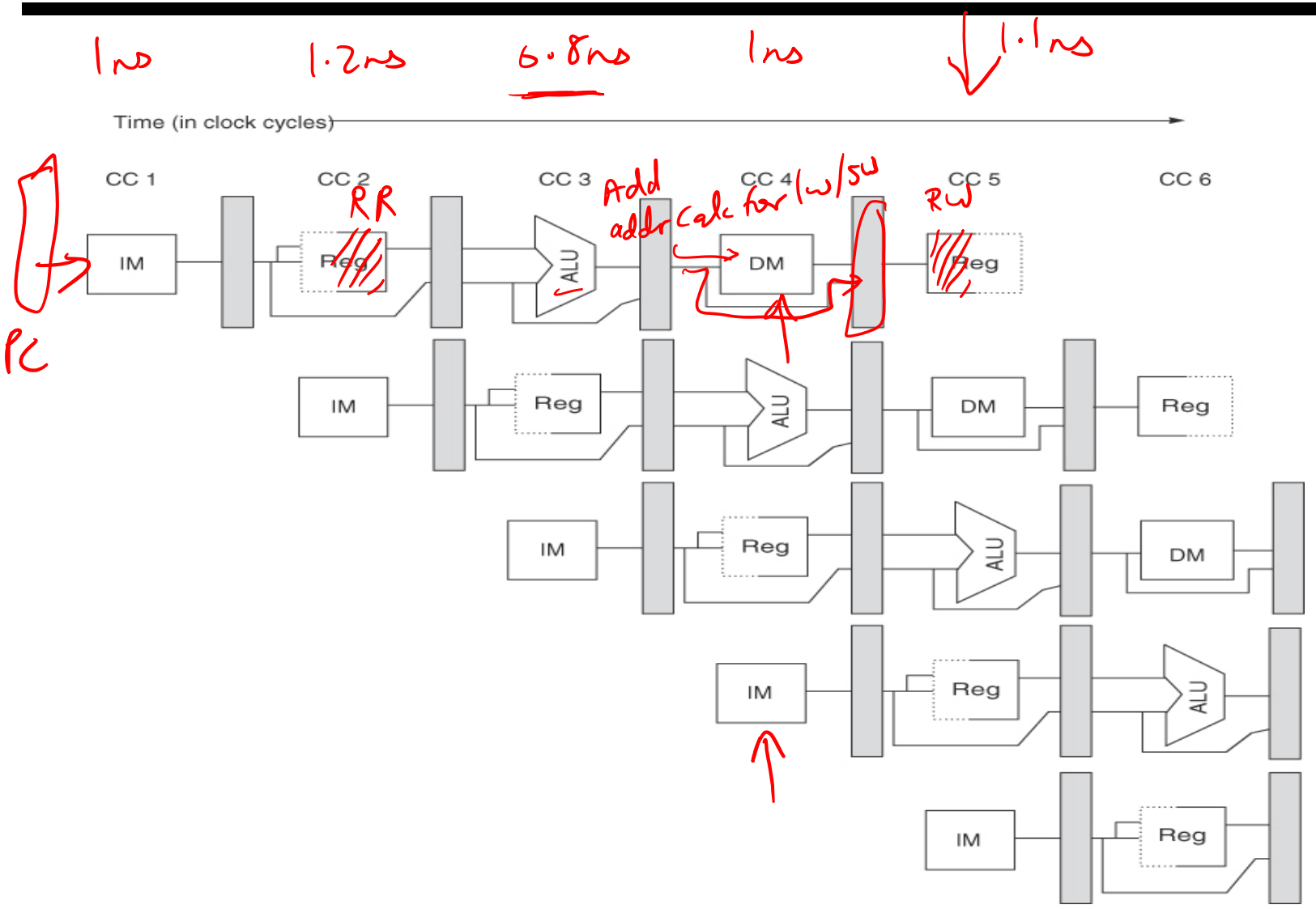
E = 140 x 666 = 920 Joules

E = 60 x 20 = 1200 J

Power and Energy

A 5-Stage Pipeline

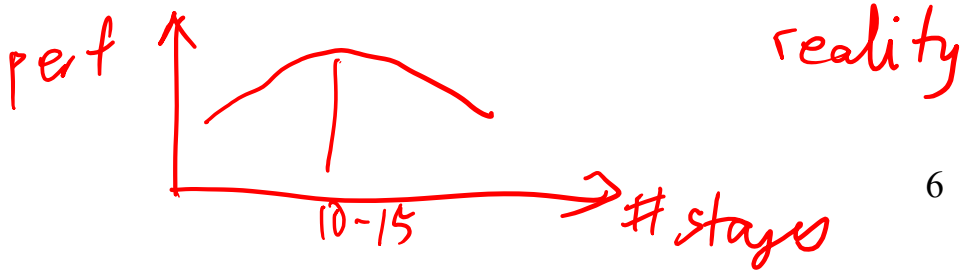
Cycle time = 1.2ns



Source: H&P textbook

Performance Improvements?

- Does it take longer to finish each individual job? Yes, latches are an additional overhead
- Does it take shorter to finish a series of jobs? Yes, because of parallelism
- 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? 10x impr, 5x impr, ideal conditions



Quantitative Effects

Easily from unpipelined to pipelined

- As a result of pipelining:
  - Time in ns per instruction goes up (bad) because of latch ovhd
  - Each instruction takes more cycles to execute (bad) defn of cycle has changed
  - But… average CPI remains roughly the same
  - Clock speed goes up (sound)
  - Total execution time goes down, resulting in lower average time per instruction (sound)
  - 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

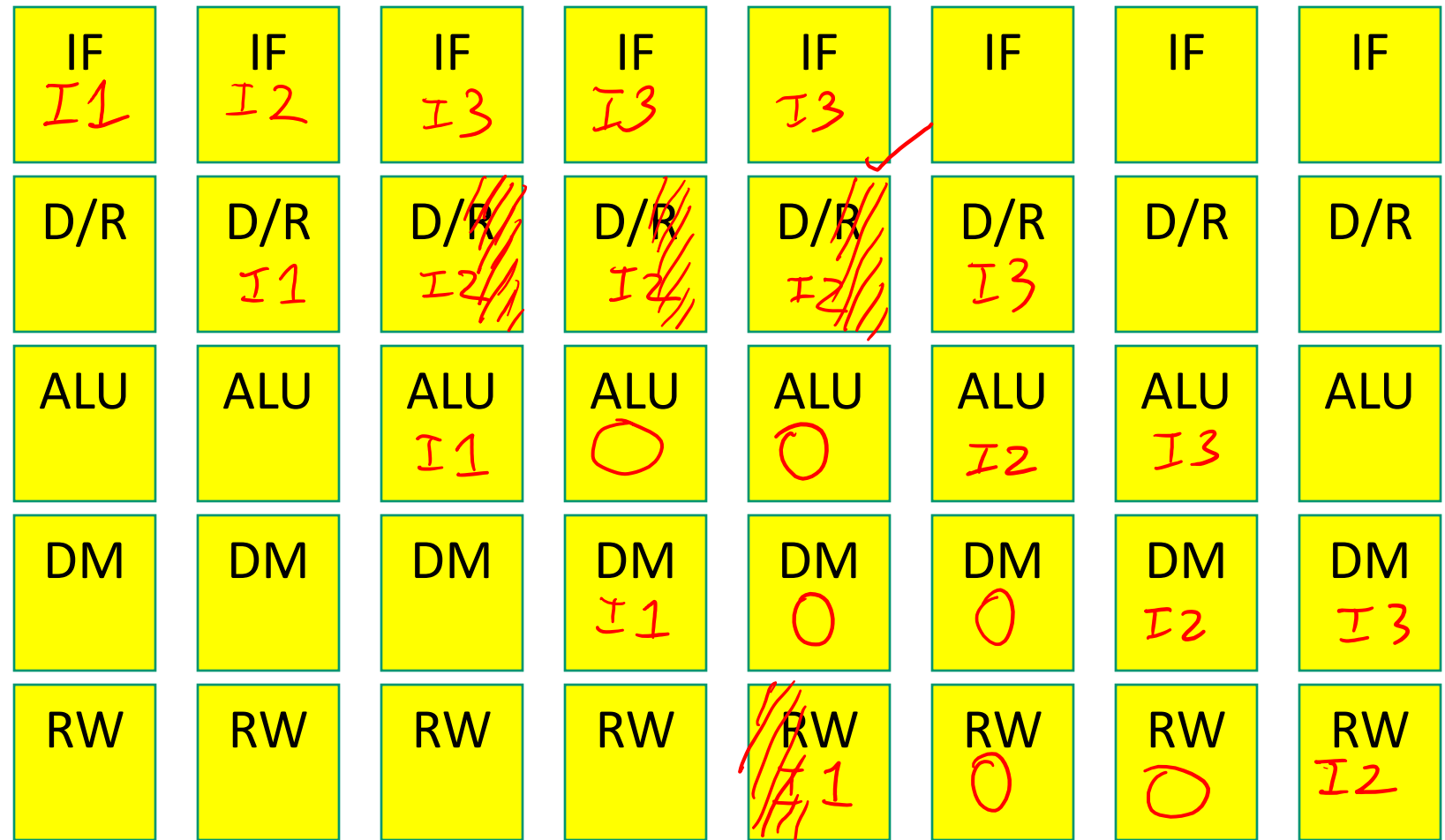
- Example: a unified instruction and data cache → stage 4 (MEM) and stage 1 (IF) can never coincide
- The later instruction and all its successors are delayed until a cycle is found when the resource is free → these are pipeline bubbles
- Structural hazards are easy to eliminate – increase the number 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



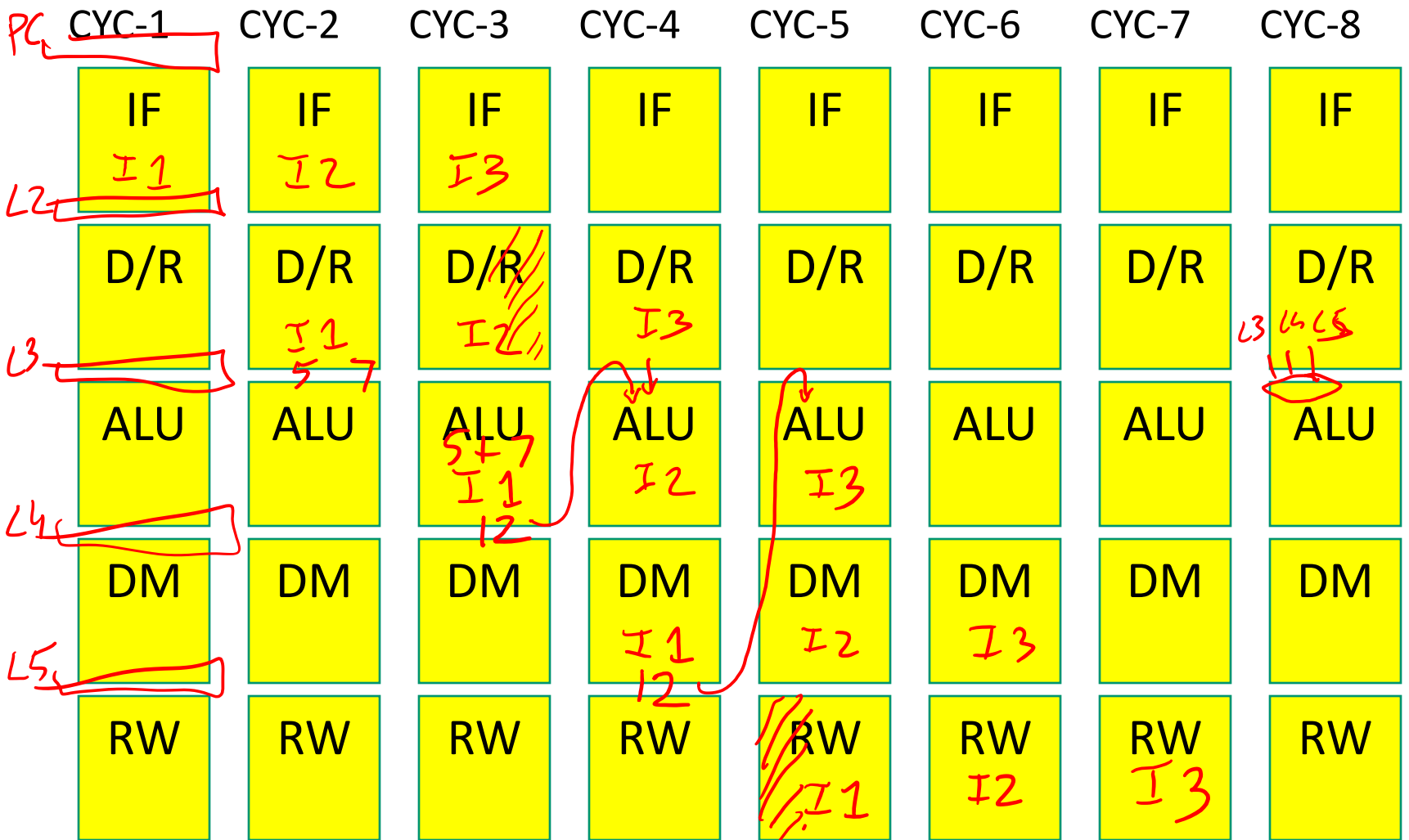
Example 1 – No Bypassing

- 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

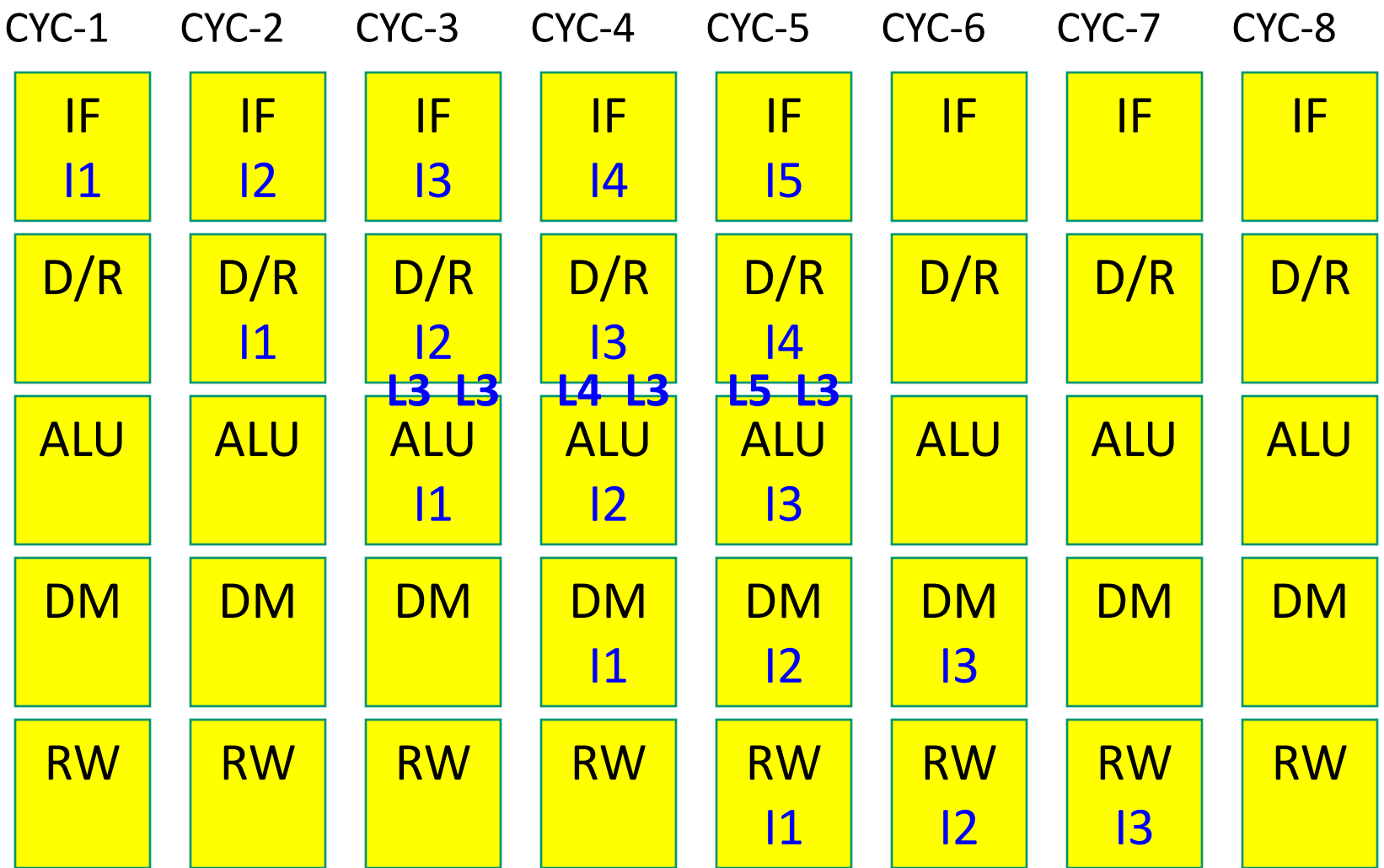
Example 2 – Bypassing

- 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.



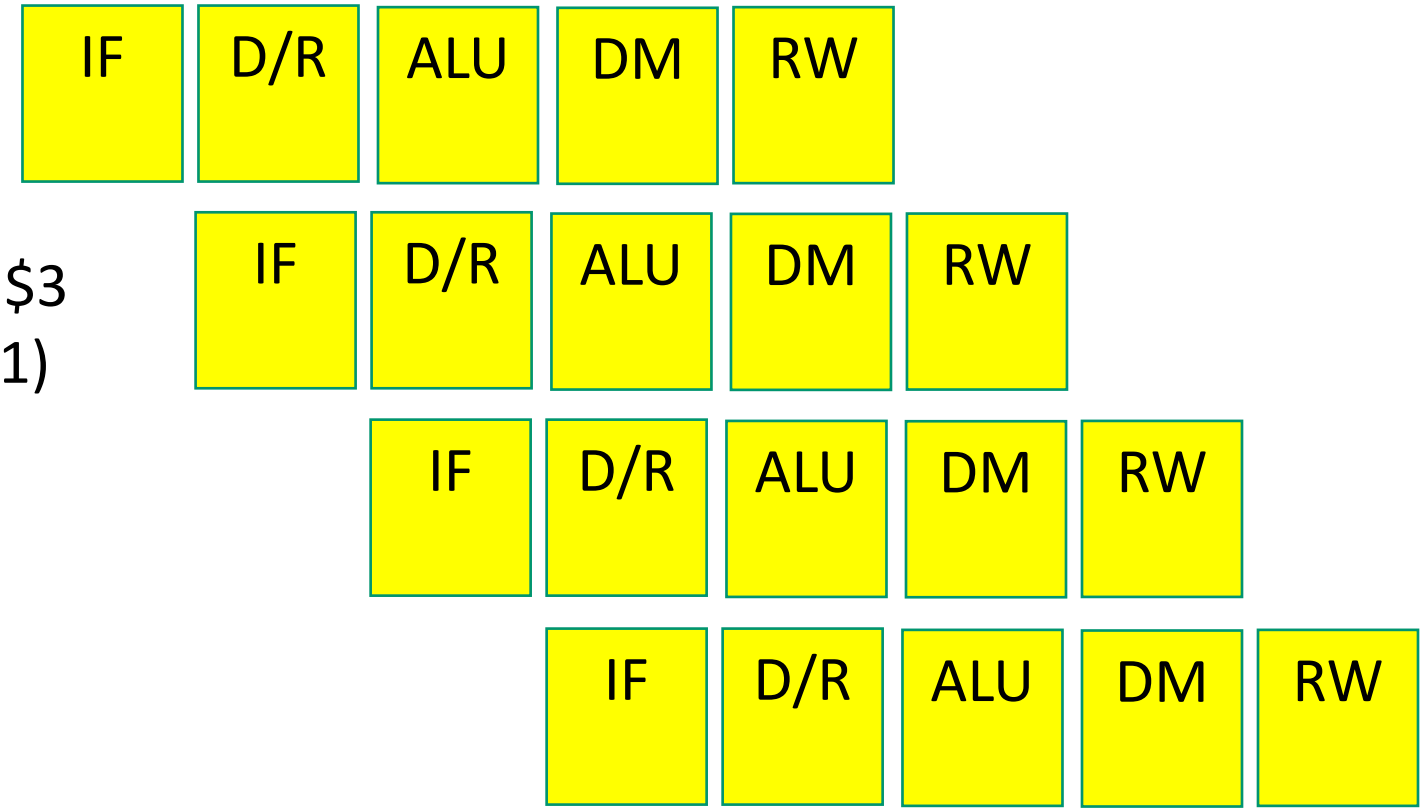
Example 2 – Bypassing

- 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.



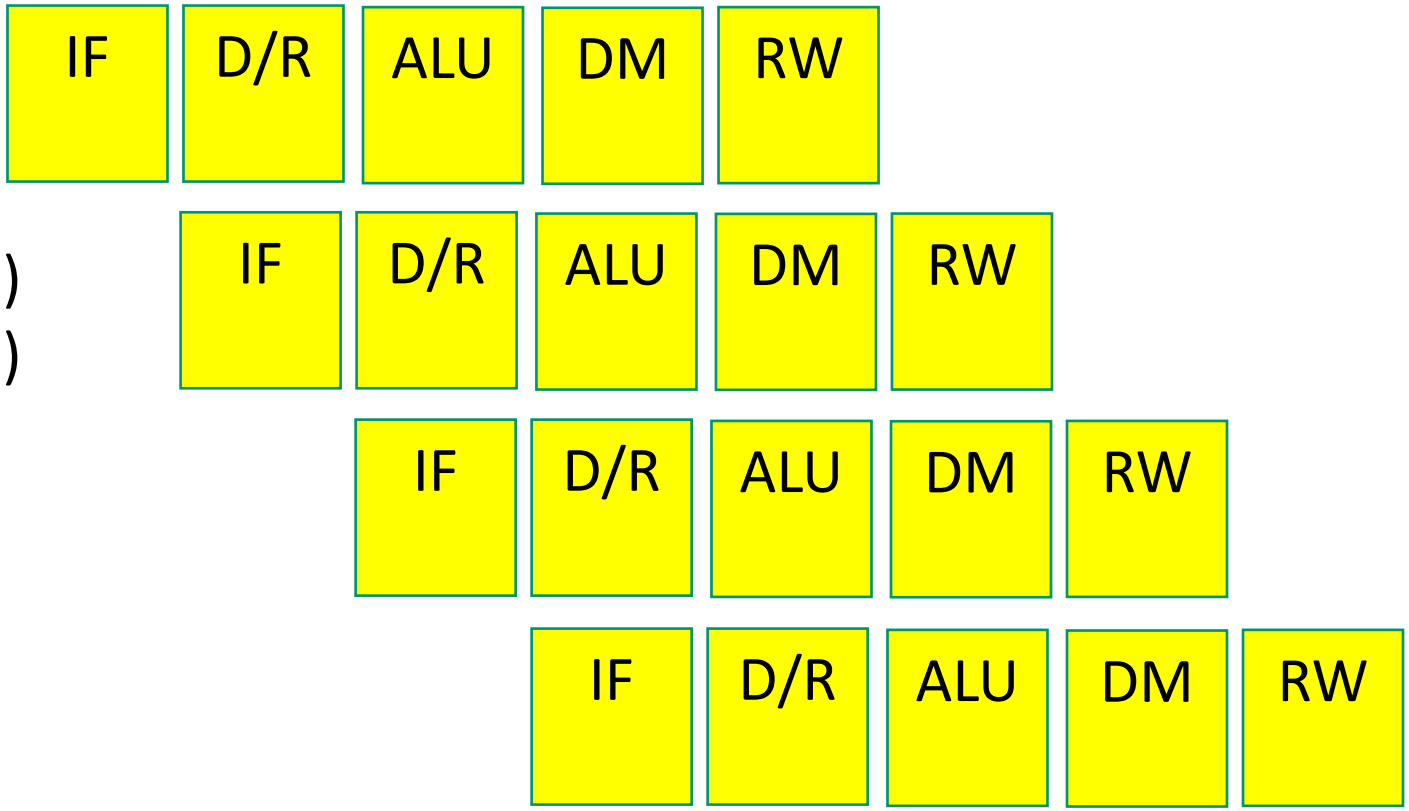
Problem 1

```
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)