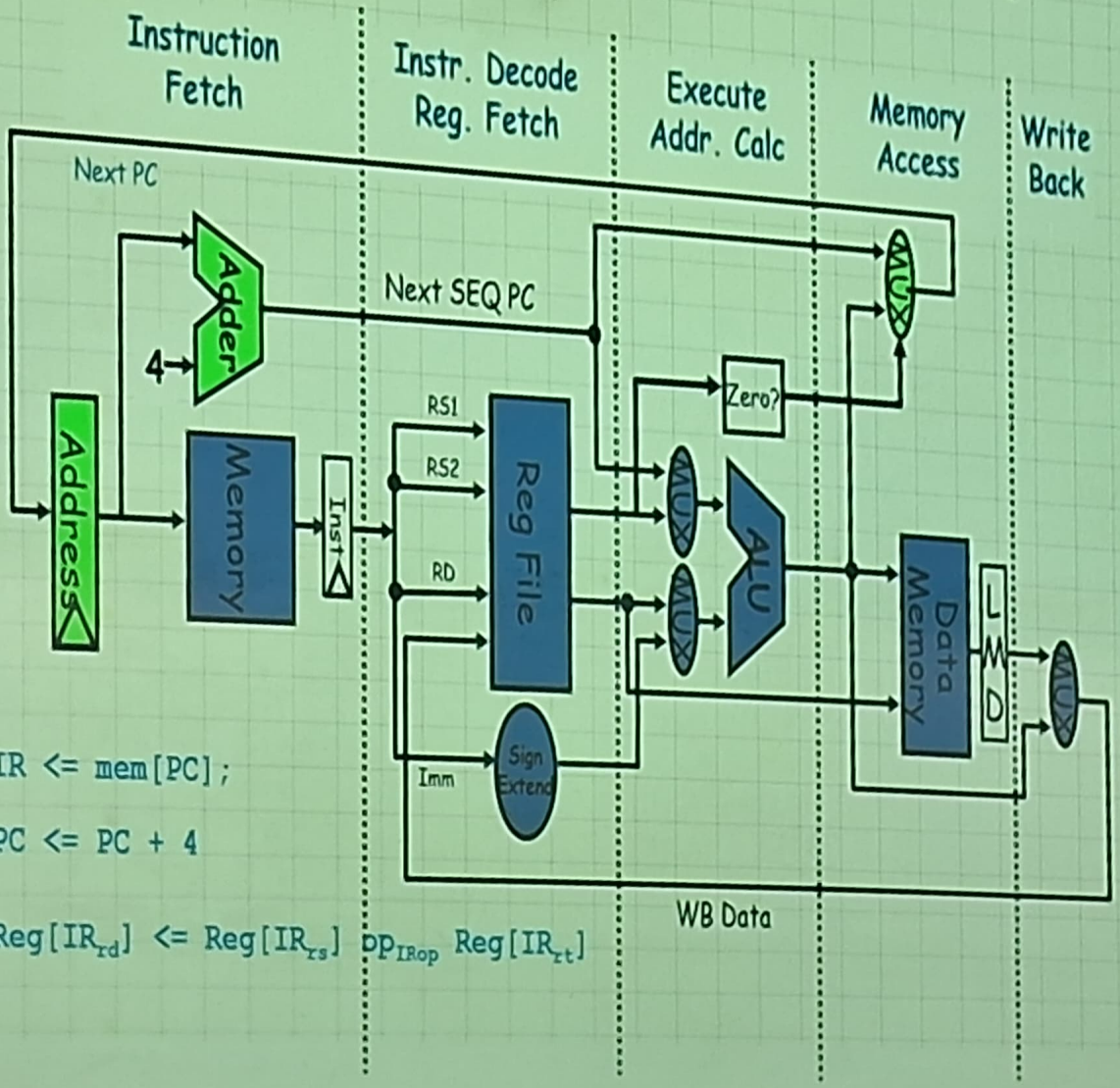
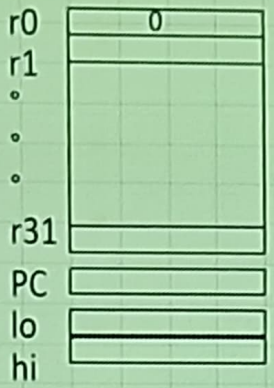
A Typical Datapath



Example: MIPS R3000



Programmable storage

- 2^32 x bytes
- 31 x 32-bit GPRs (R0=0)
- 32 x 32-bit FP regs (paired DP)
- HI, LO, PC

Data types ?

Format ?

Addressing Modes?

where is mul/div

Arithmetic logical

Add, AddU, Sub, SubU, And, Or, Xor, Nor, SLT, SLTU,
AddI, AddIU, SLTI, SLTIU, AndI, OrI, XorI, *LUI*
SLL, SRL, SRA, SLLV, SRLV, SRAV

Memory Access

LB, LBU, LH, LHU, LW, LWL,LWR
SB, SH, SW, SWL, SWR

Control

J, JAL, JR, JALR
BEq, BNE, BLEZ,BGTZ,BLTZ,BGEZ,BLTZAL,BGEZAL

why so many variants

32-bit instructions on word boundary

The Memory Abstraction

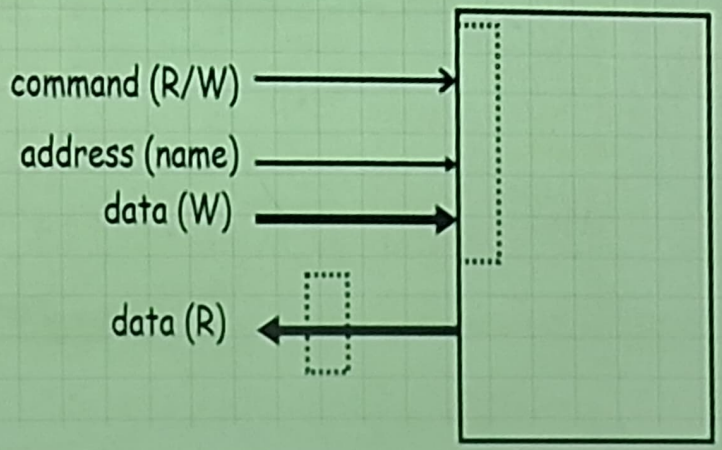
- Association of <name, value> pairs
  - typically named as byte addresses
  - often values aligned on multiples of size

Size of ( )

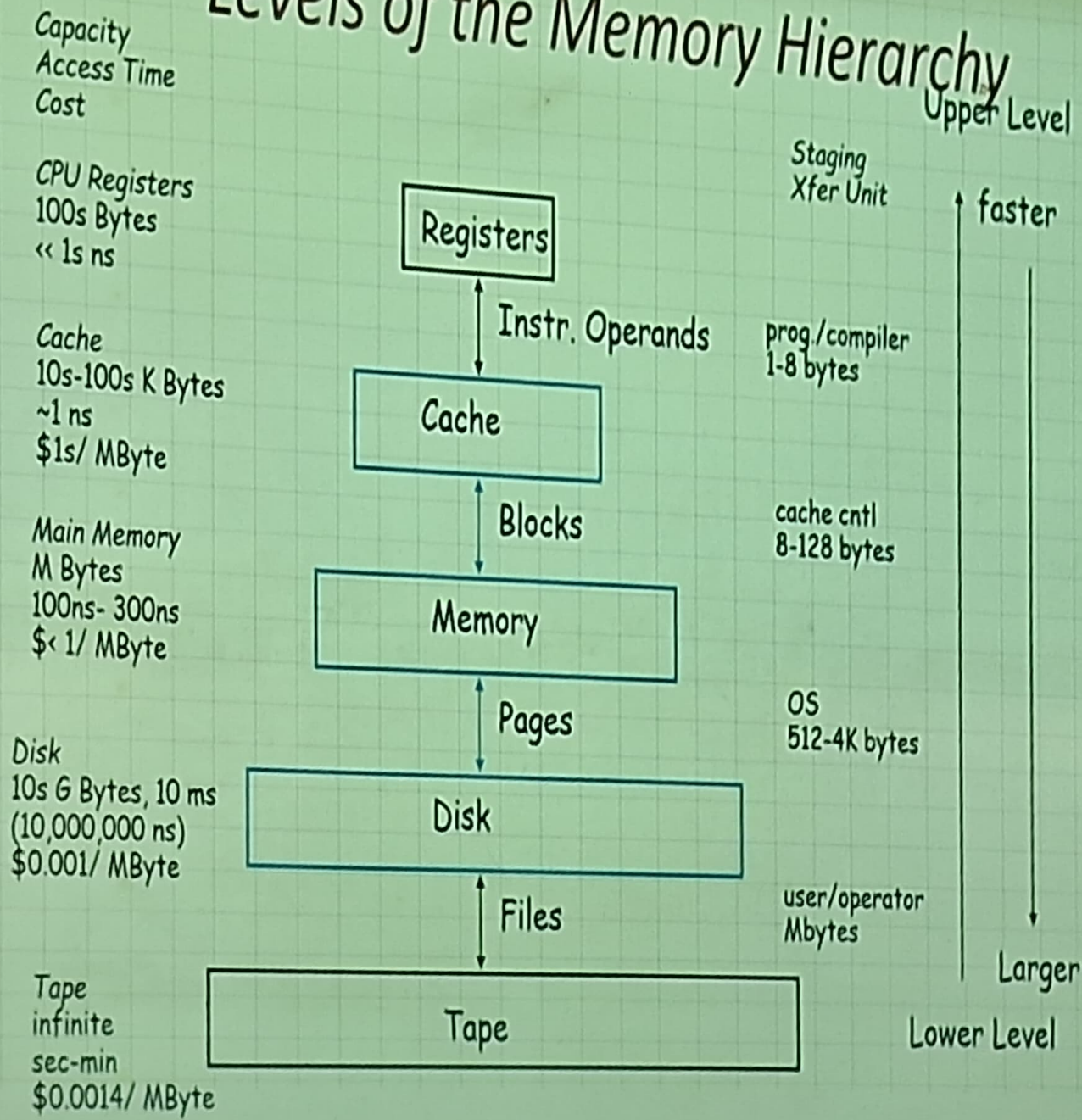
Char a[100]

myData xyz[100]

Struct {
int a;
Char b;
} myData;



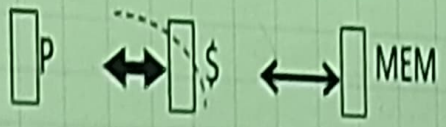
Levels of the Memory Hierarchy

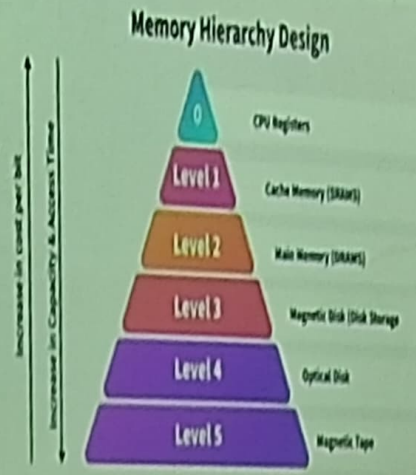


Principle of Locality:

- Program access a relatively small portion of the address space at any instant of time.

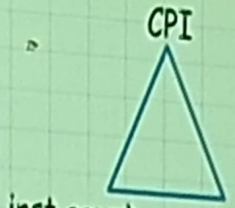
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)
- Last 30 years, HW relied on locality for speed





Processor performance equation

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$



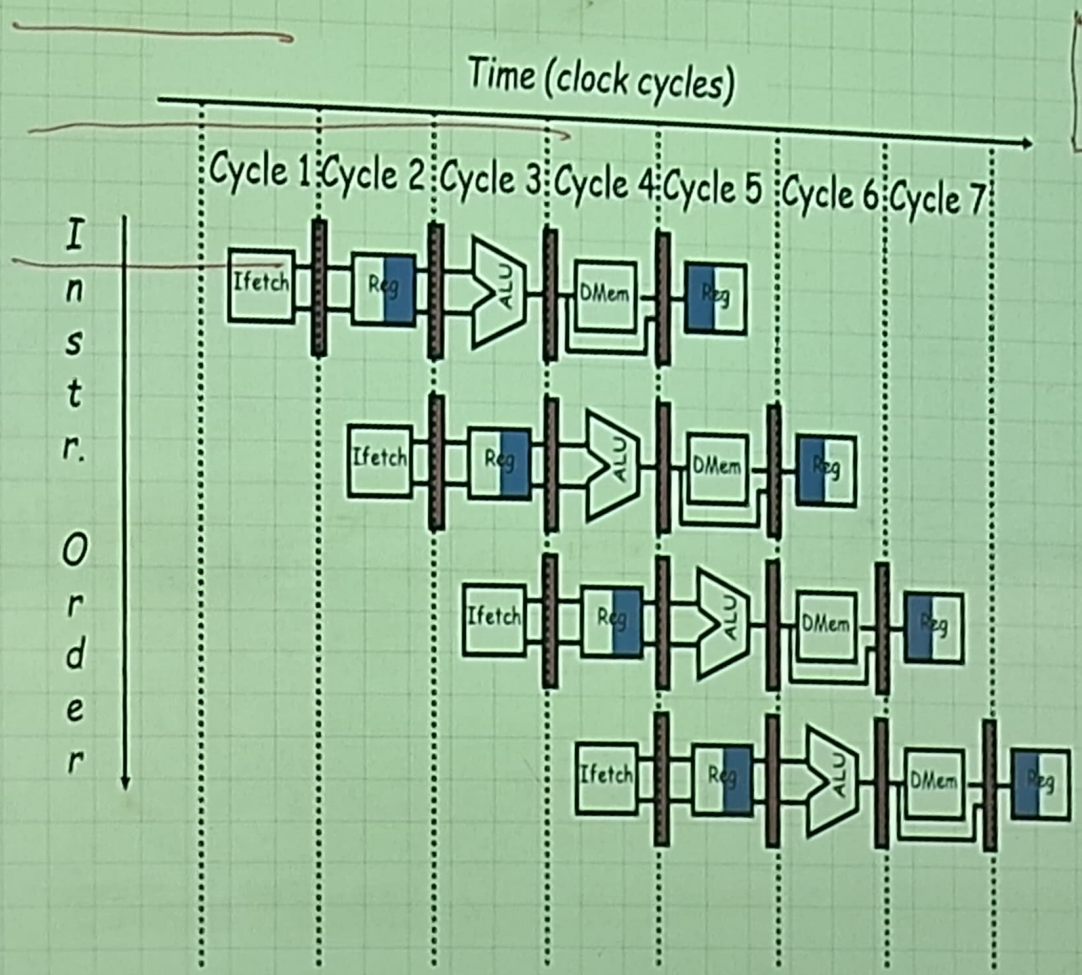
	Inst Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Inst. Set.	X	X	
Organization		X	X
Technology			X

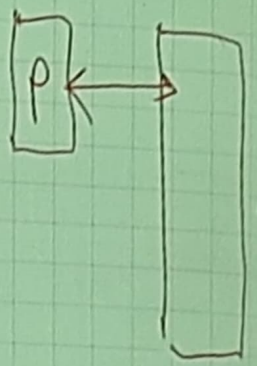
mem [x]

R1 op [+/-] mem [y]

ld R2, mem(x)
add R3, R2, R1
st R3, mem[y]

Pipelined Instruction Execution





Memory

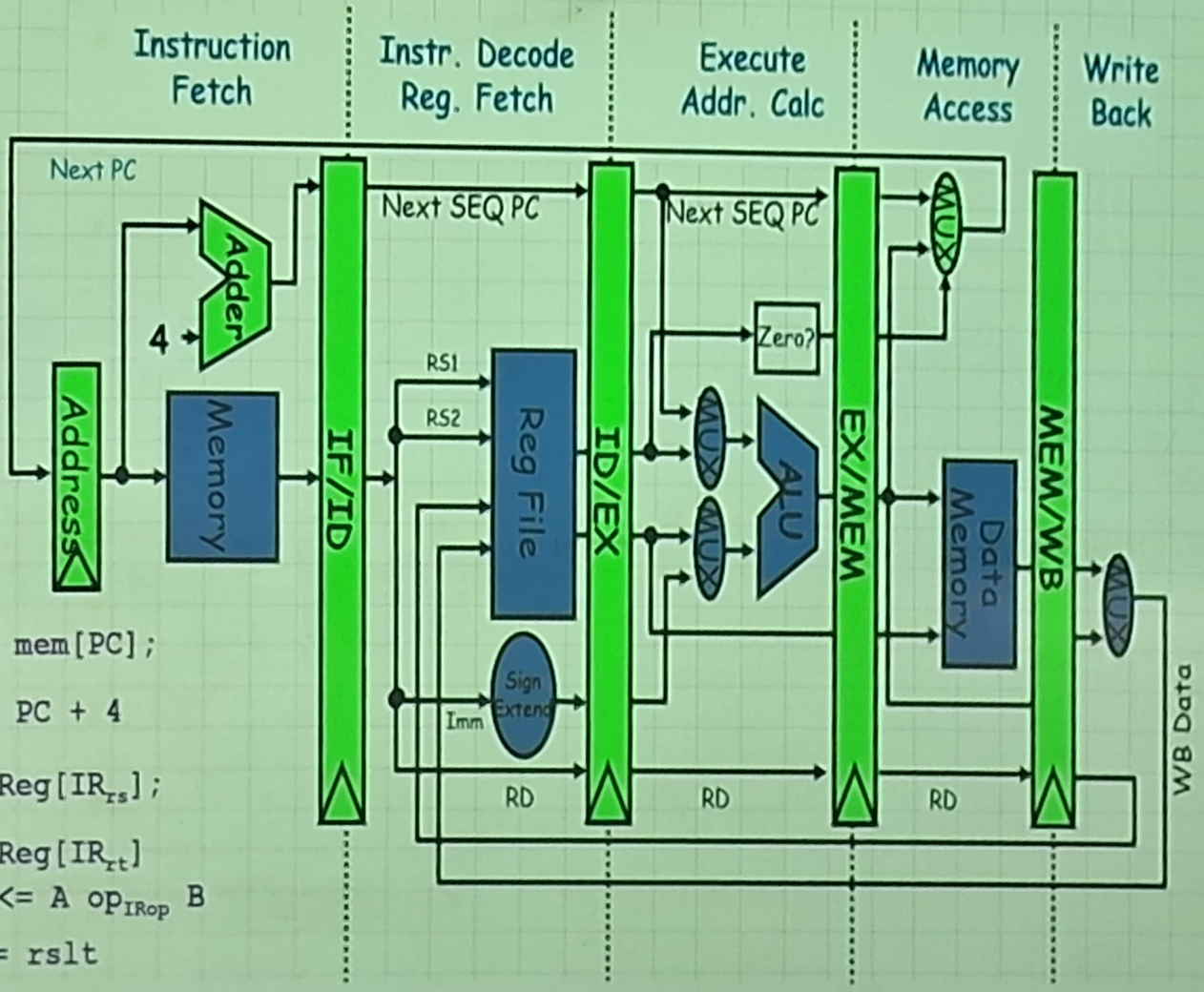
b = 6

a = 3

a = b + 3

a = 7

Introduce Pipeline Registers



IR <= mem[PC];

PC <= PC + 4

A <= Reg[IR_{rs}];

B <= Reg[IR_{rt}]

rslt <= A op_{IRop} B

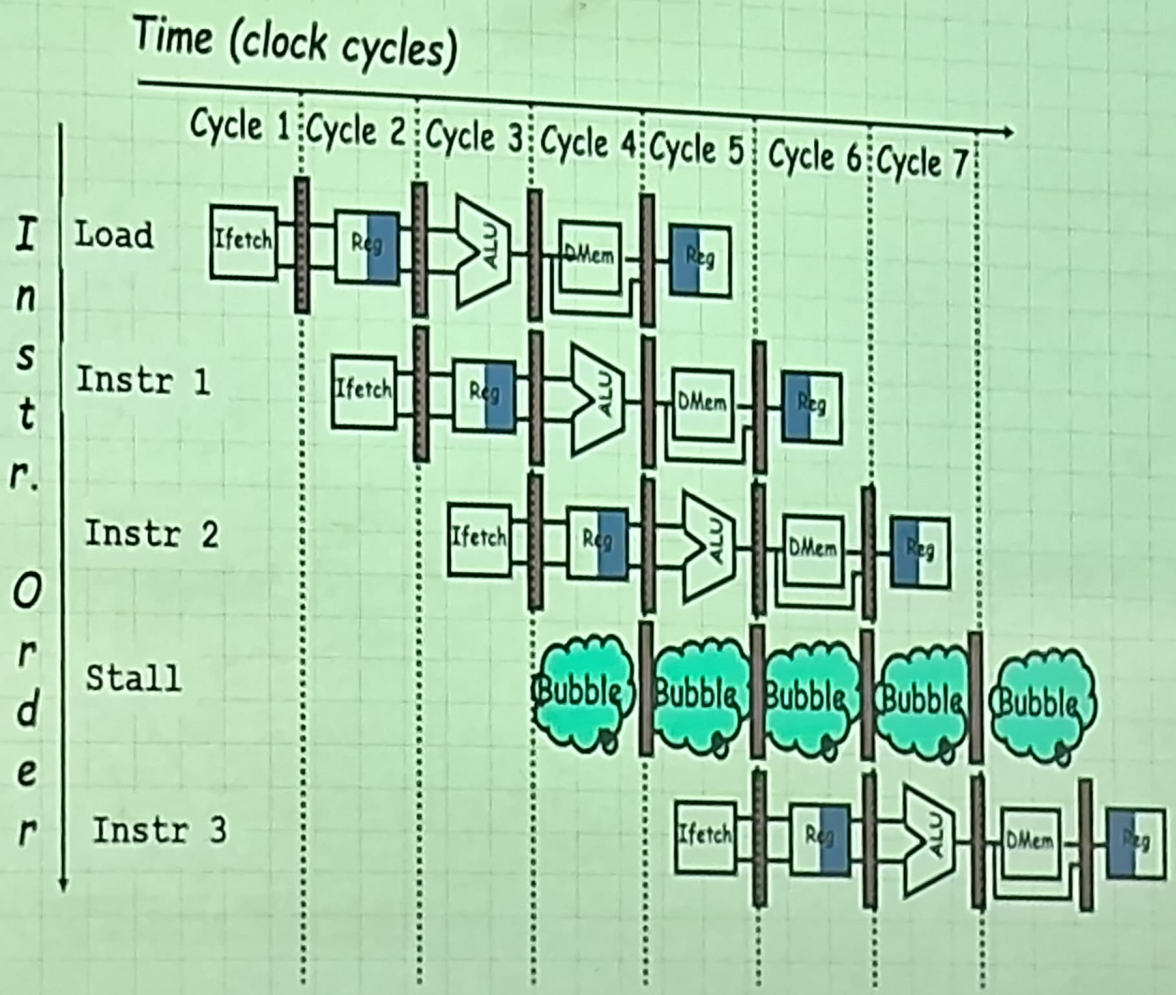
WB <= rslt

Reg[IR_{rd}] <= WB

Pipelining is not quite that easy!

- Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: HW cannot support this combination of instructions (single person to fold and put clothes away)
  - Data hazards: Instruction depends on result of prior instruction still in the pipeline (missing sock)
  - Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).

One Memory Port/Structural Hazards



Speed Up Equation for Pipelining

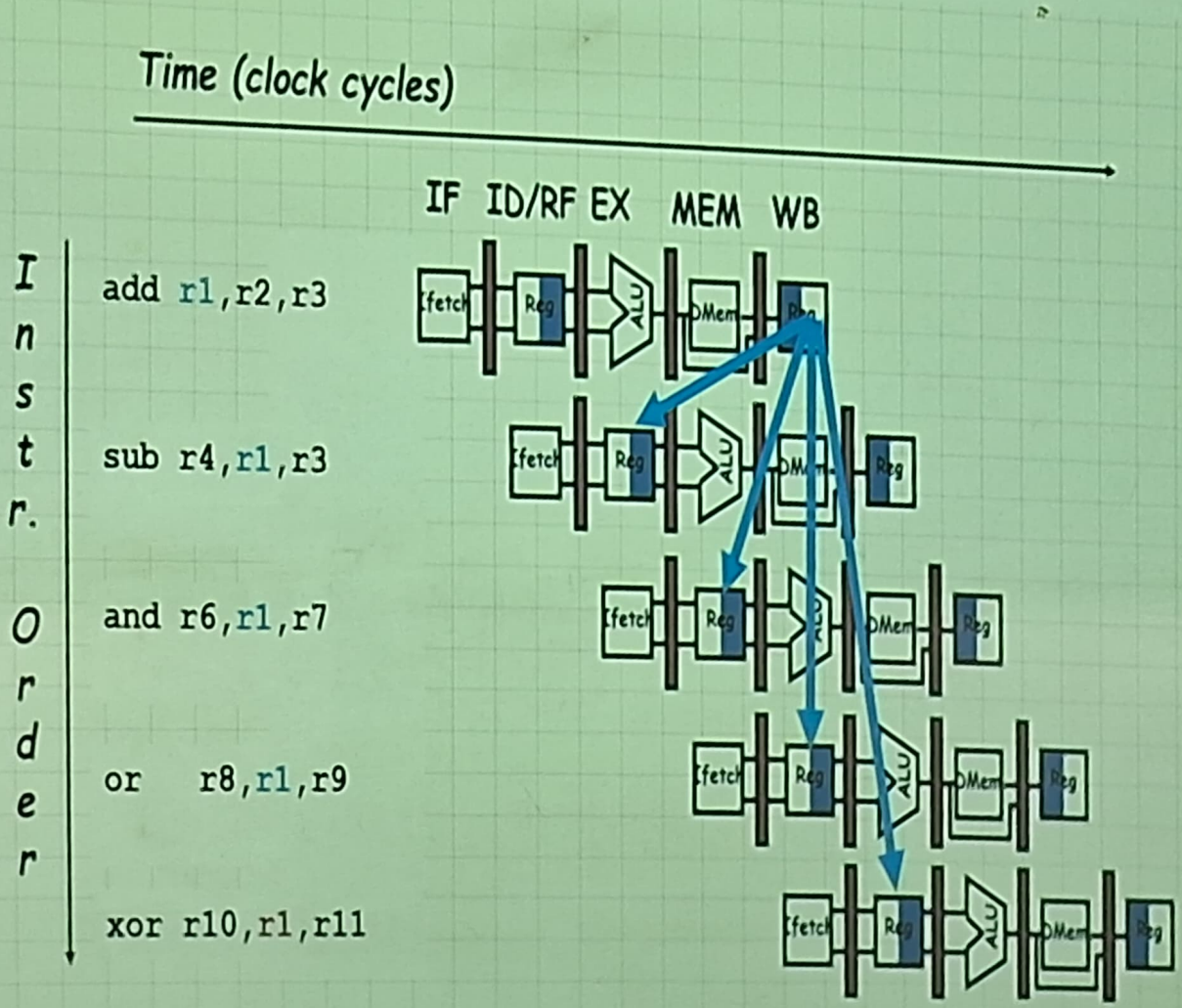
$$CPI_{pipelined} = \text{Ideal CPI} + \text{Average Stall cycles per Inst}$$

$$\text{Speedup} = \frac{\text{Ideal CPI} \times \text{Pipeline depth}}{\text{Ideal CPI} + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{unpipelined}}{\text{Cycle Time}_{pipelined}}$$

For simple RISC pipeline, CPI = 1:

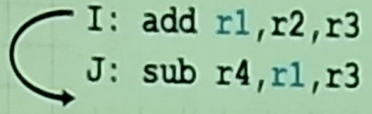
$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{unpipelined}}{\text{Cycle Time}_{pipelined}}$$

Data Hazard on R1



Three Generic Data Hazards

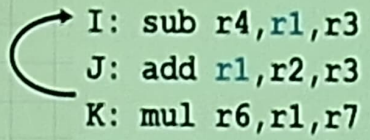
- Read After Write (RAW)
  Instr$_J$ tries to read operand before Instr$_I$ writes it



- Caused by a "Dependence" (in compiler nomenclature)
  This hazard results from an actual need for communication.

Three Generic Data Hazards

- Write After Read (WAR)
  $Instr_J$ writes operand *before* $Instr_I$ reads it

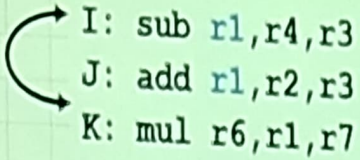


- Called an "anti-dependence" by compiler writers.
  This results from reuse of the name "r1".

- Can't happen in simple RISC 5 stage pipeline because:
  - All instructions take 5 stages, and
  - Reads are always in stage 2, and
  - Writes are always in stage 5

Three Generic Data Hazards

- Write After Write (WAW)
  $Instr_J$ writes operand *before* $Instr_I$ writes it.



- Called an "output dependence" by compiler writers
  This also results from the reuse of name "r1".
- Can't happen in Simple 5 stage pipeline because:
  - All instructions take 5 stages, and
  - Writes are always in stage 5
- Will see WAR and WAW in more complicated pipes

Forwarding to Avoid Data Hazard