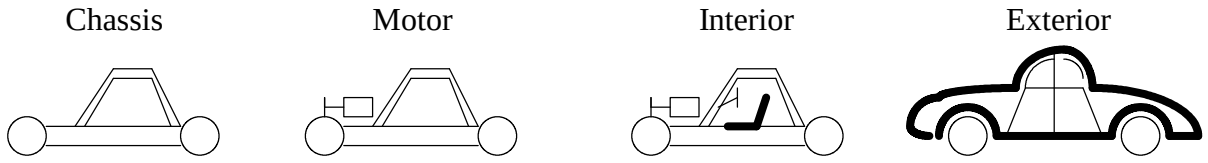
Instruction Pipelining - assembly-line idea used to speed instruction completion rate

Assume that an automobile assembly process takes 4 hours.



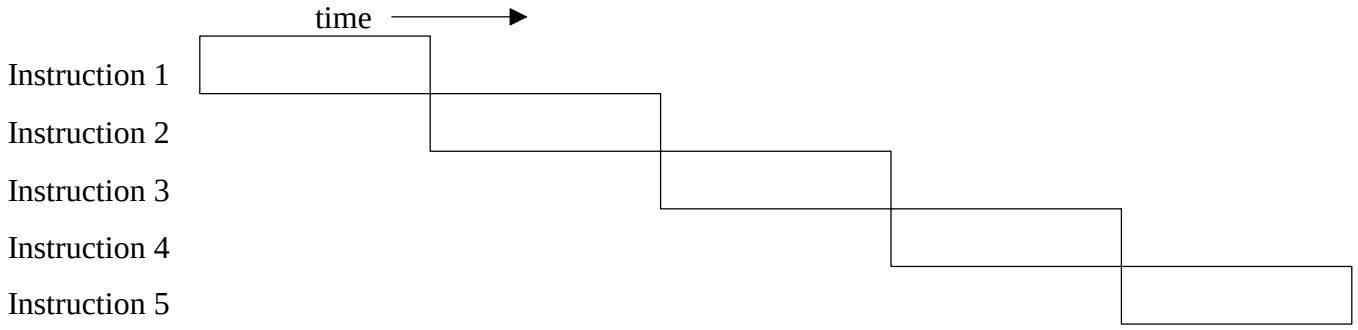
If you divide the process into four equal stages, then ideally

$$\text{time between completions} = \frac{\text{time to complete one car}}{\text{\# of stages}}$$

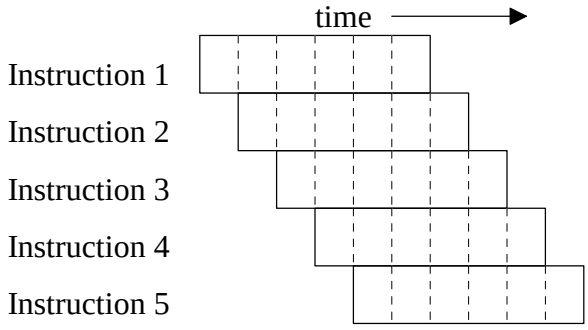
Problems:

- stages might not be balanced
- overhead of moving cars between stages
- two stages need same specialized tool (structural hazard)

Serial Execution



Pipelined Execution - goal is to complete one instruction per clock cycle



MIPS (simple RISC) instruction formats:

all instruction 32-bits in length

Arithmetic: add R1, R2, R3

opcode	dest reg	operand 1 reg	operand 2 reg	unused

Unconditional Branch/"jump": j someLabel

opcode	large offset from PC or absolute address

Arithmetic with immediate: addi R1, R2, 8

opcode	operand 1 reg	operand 2 reg	immediate value

Conditional Branch: beq R1, R2, end_if

opcode	operand 1 reg	operand 2 reg	PC-relative offset to label

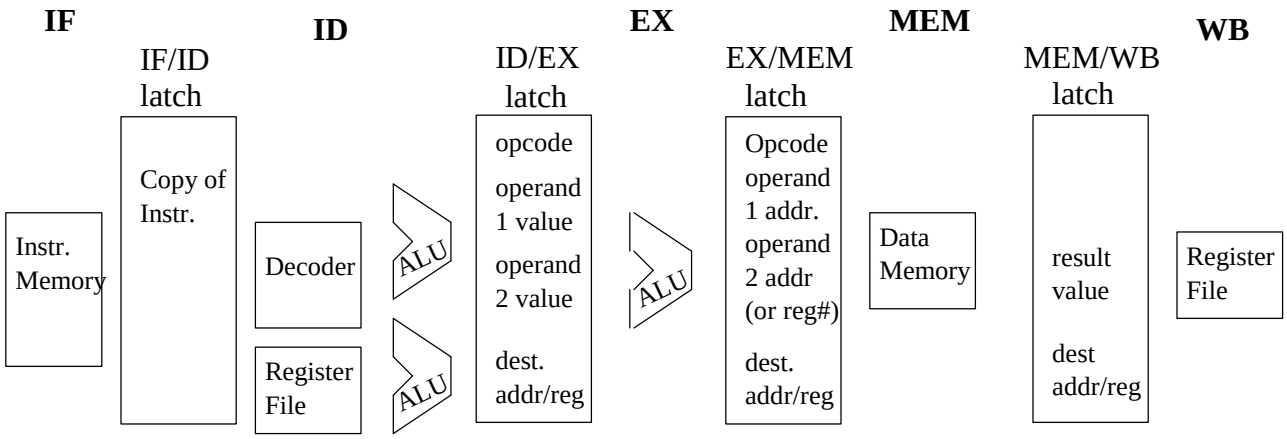
Load/Store: lw R1, 16(R2)

opcode	operand reg	base reg	offset from base reg

RISC Instruction Pipelining Example: One possible break down of instruction execution**.**

Stage	**Abbreviation**	**Actions**
Instruction Fetch	IF	Read next instruction into CPU and increment PC by 4 byte (to next instruction)
Instruction Decode	ID	Determine opcode, read registers, compare registers (if branch), sign-extend immediate if needed, compute target address of branch, update PC if branch
Execution / Effective addr	EX	Calculate using operands prepared in ID ▪ memory ref: add base reg to offset to form effective address ▪ reg-reg ALU: ALU performs specified calculation ▪ reg-immediate ALU: ALU performs specified calculation
Memory access	MEM	▪ load: read memory from effective address into pipeline register ▪ store: write reg value from ID stage to memory at effective address
Write-back	WB	▪ ALU or load instruction: write result into register file

Pipeline latches/registers between each stage. Hold temporary results and act like an IR. Some of the hardware components used (e.g., Memory and Register File) are shown as if they are duplicated, but they are not.



Problems that delay/*stall* the pipeline:

- ***structural hazard*** **-** a piece of hardware is needed by several stages at the same time, e.g., Memory in IF, and MEM. This might require stages to sequentially access the hardware, or duplicate into two memories.

- ***data hazard*** - an instruction depends on the results of a previous instruction which has not been calculated yet. (RAW) read-after-write example: ADD R3, R2, R1 ; R3← R2 + R1

  SUB R4, R3, R5 ; R4 ← R3 - R5

  In what stage does the ADD instruction update R3?

  In what stage does the SUB instruction read R3?

- ***control hazard*** - branching makes it difficult to fetch the "correct" instructions to be executed

Data Hazards:

Wrong result in below since SUB read the "old" value of R3 in ID, before ADD updates R3 in WB stage.

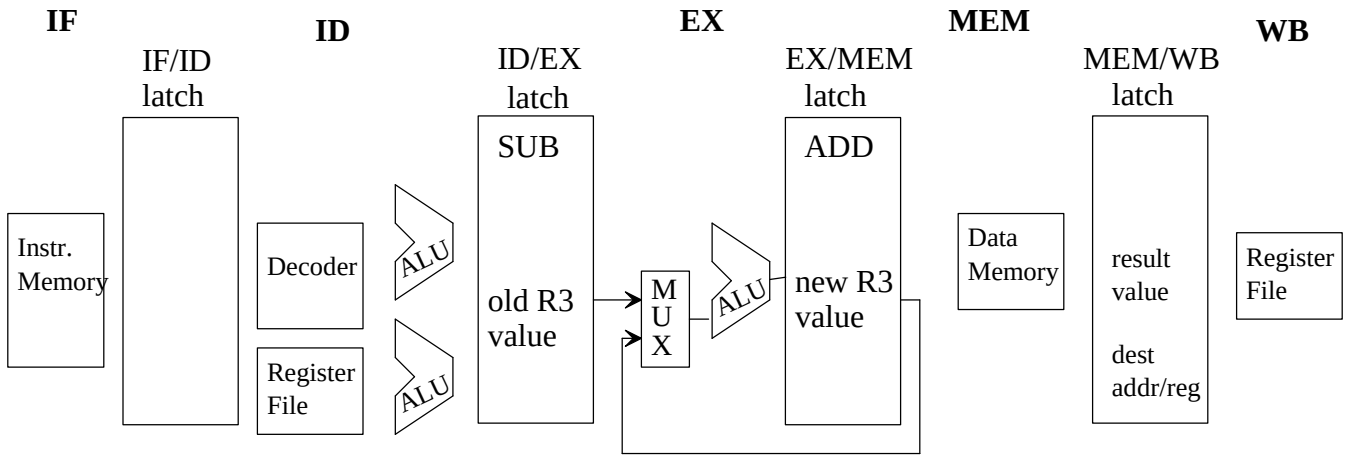
	Time →											
Instructions	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
ADD R3, R2, R1	IF	ID	EX	MEM	**WB**							
SUB R4, R3, R5		IF	**ID**	EX	MEM	WB						

Solution Alternatives:

1) Introduce stalls - stall reading of R3 in last half of ID until ADD writes R3 in first half of WB

	Time →											
Instructions	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
ADD R3, R2, R1	IF	ID	EX	MEM	**WB**							
SUB R4, R3, R5		IF	stall	stall	**ID**	EX	MEM	WB				

2) Add additional hardware (***bypass-signal paths***) to "**foward**" R3's new value to the SUB instruction:

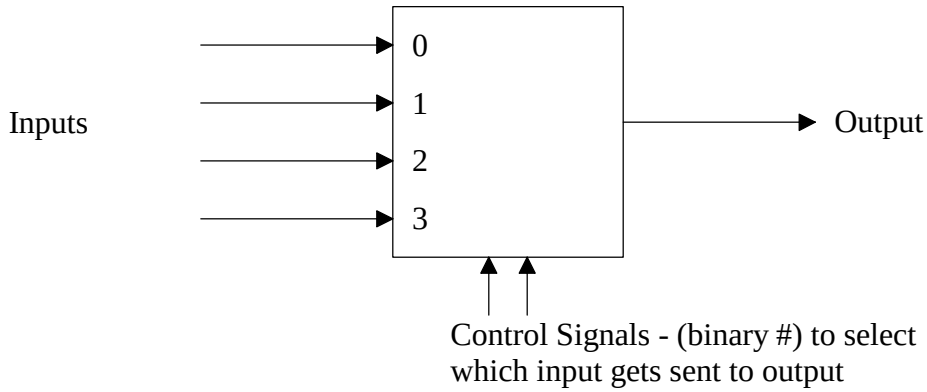


No stalls needed in this case.

	Time →											
Instructions	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
ADD R3, R2, R1	IF	ID	**EX**	MEM	WB							
SUB R4, R3, R5		IF	ID	**EX**	MEM	WB						

What would control the MUX?

MUX Operation:



Consider the following code: ADD R3, R2, R1
STORE R3, 4(R4)

What would the timing be **without** bypass-signal paths/forwarding?

(Assume that R3 can be written in the first half of the WB stage and its new value read in the last half of the same stage).

	Time →											
Instructions	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
ADD R3, R2, R1	IF	ID	EX	MEM	WB							
STORE R3, 4(R4)		IF										

Solution:

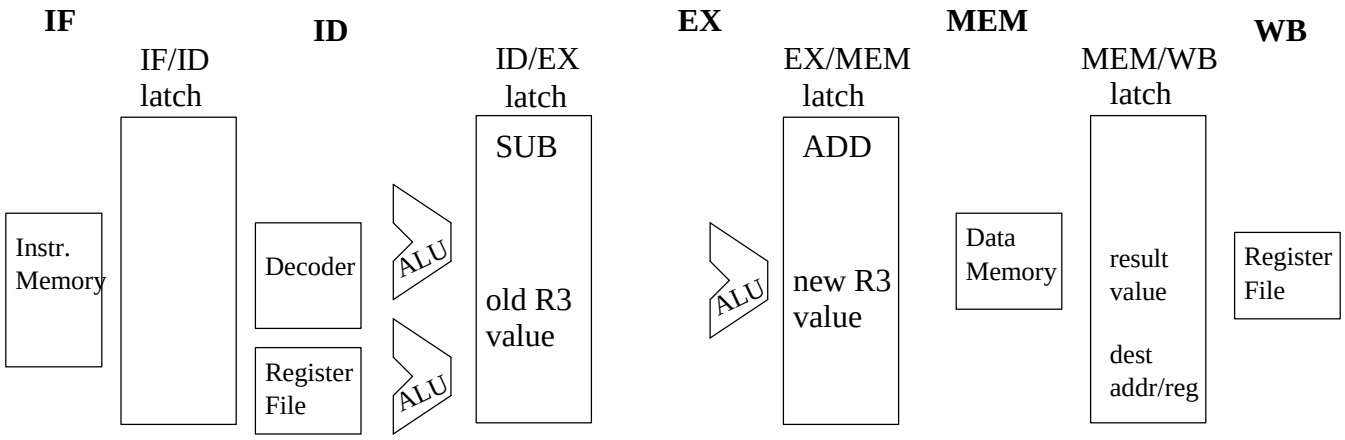
	Time →											
Instructions	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
ADD R3, R2, R1	IF	ID	EX	MEM	**WB**							
STORE R3, 4(R4)		IF	stall	stall	**ID**	EX	MEM	WB				

What would the timing be with bypass-signal paths?

	Time →											
Instructions	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
ADD R3, R2, R1	IF	ID	EX	MEM	WB							
STORE R3, 4(R4)		IF										

	Time →											
Instructions	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
ADD R3, R2, R1	IF	ID	**EX**	MEM	WB							
STORE R3, 4(R4)		IF	ID	EX	**MEM**	WB						

What (draw) bypass-signal paths are needed for the above example.



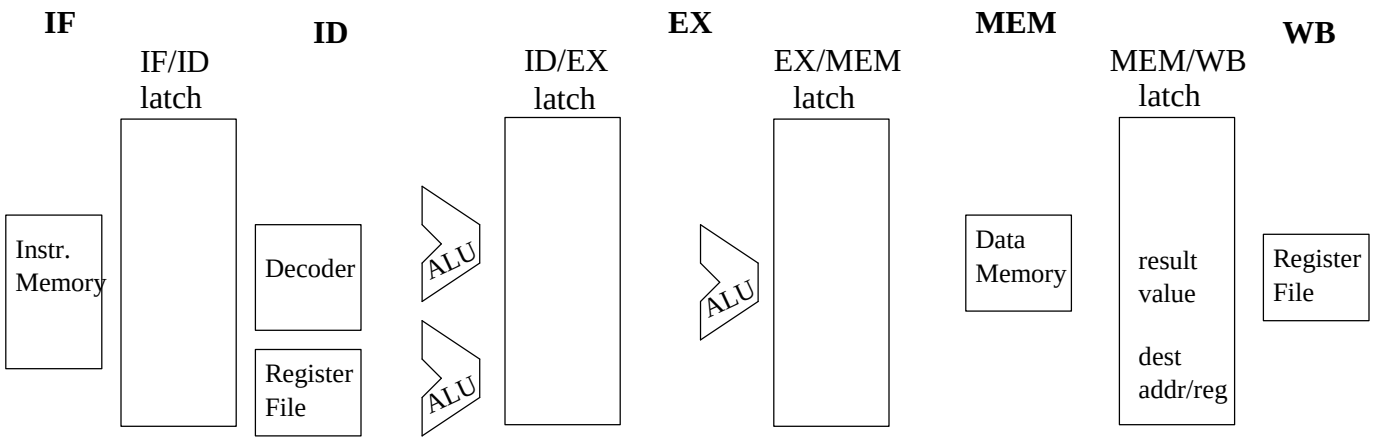
How many cycles are needed to perform the following AL program **without** forwarding?

	Time →															
Instructions	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
ADD R3, R2, R1	IF	ID	EX	MEM	WB											
SUB R5, R4, R3		IF														
ADD R6, R3, R2																

How many cycles are needed to perform the following AL program **with** forwarding?

	Time →															
Instructions	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
ADD R3, R2, R1	IF	ID	EX	MEM	WB											
SUB R5, R4, R3		IF														
ADD R6, R3, R2																

Draw ALL the bypass-signal paths needed for the above example.



How many cycles are needed to perform the following AL program **without** forwarding?

	Time →															
Instructions	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
ADD R3, R2, R1	IF	ID	EX	MEM	WB											
LOAD R4, 4(R3)		IF														
SUB R5, R4, R3																
STORE R5, 8(R6)																
ADD R6, R5, R4																

How many cycles are needed to perform the following AL program **with** forwarding?

	Time →															
Instructions	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
ADD R3, R2, R1	IF	ID	EX	MEM	WB											
LOAD R4, 4(R3)		IF														
SUB R5, R4, R3																
STORE R5, 8(R6)																
ADD R6, R5, R4																

Draw ALL the bypass-signal paths needed for the above example.