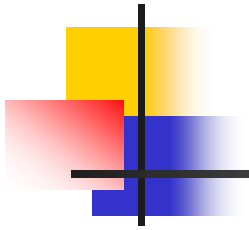
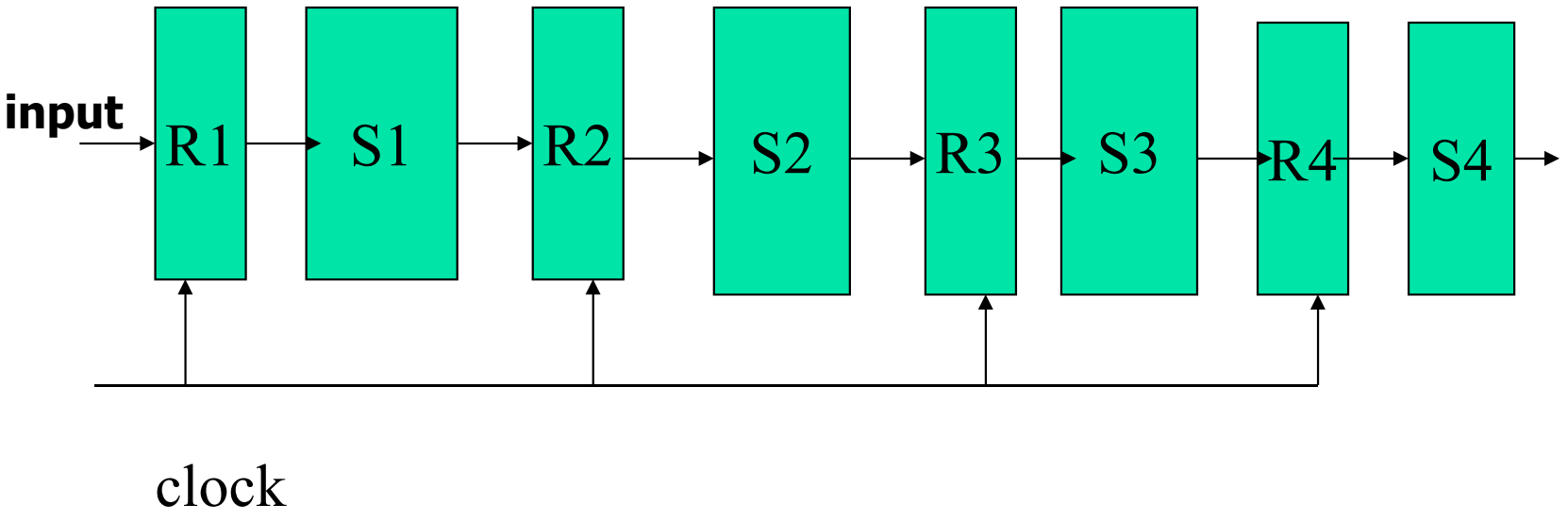
Pipelining



Pipelining is a technique of decomposing a sequential process in to sub operations, with each sub operation being executed in a separate dedicated segment that operates concurrently with all other segments.

Linear Pipeline



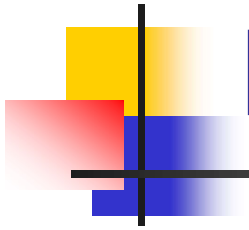
Pipeline Characteristic diagram

	1	2	3	4	5	6	7	8	9	10	11
T	T1	T2	T3	T4	T5	T6	T7	T8			
A		T1	T2	T3	T4	T5	T6	T7	T8		
S			T1	T2	T3	T4	T5	T6	T7	T8	
K				T1	T2	T3	T4	T5	T6	T7	T8
S											

Non-pipeline processor characteristic diagram

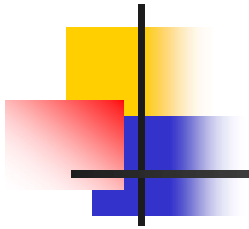
- Clock cycles→

TASKS	1	2	3	4	5	6	7	8	9	10	11
	T_1				T_2				T_3		
		T_1				T_2				T_3	
			T_1				T_2				T_3
				T_1				T_2			



Pipeline performance

- Let there are k- segments in a pipeline
- There are n tasks to be performed
- Let the cycle time is tp
- No of cycles needed to get the first output from the pipeline is k-cycles
- The time needed for the first task is ktp
- The remaining tasks can be performed in (k-1) cycles.



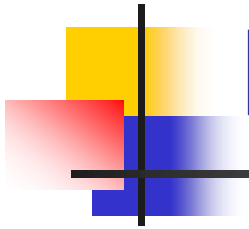
Pipeline performance …

- Thus the number of cycles needed for n-tasks is (k+n-1)
- The time needed for performing n-tasks using the pipelined processor

  (k+n-1) * tp

  the time taken by a non-pipelined processor

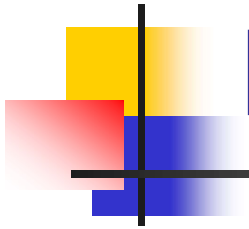
  to perform a task is tn



Pipeline performance…

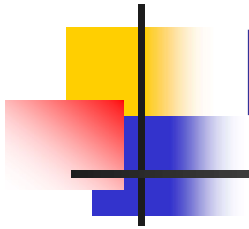
- Time taken for completing the n tasks by a non-pipelined processor is ntn
- The speedup of pipelined processor over non-pipelined processor is
- Speed up S = ntn / (k+n-1)tp
- When n is very large then the term (k+n-1)

  Is approaches to n.



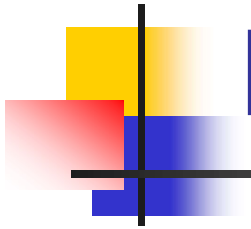
Pipeline performance …

- Thus the above equation becomes
- S=tn/ tp
- If we assume that the cycle time of pipeline and non-pipelined processor is tp
- Then tn=k tp
- S=k this shows the max speedup that can be provided by the pipeline processor is equal to the depth of the pipeline.



MIPS pipeline

- Instruction Fetch cycle (IF)
- Instruction decode/Register fetch (ID)
- Execution/Effective address cycle (EX)
- Memory access/Branch completion (MEM)
- Write-Back cycle (WB)



Pipeline char diag

Instr Num	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
instr i	IF	ID	EX	MEM	WB				
instr i+1		IF	ID	EX	MEM	WB			
instr i+2			IF	ID	EX	MEM	WB		
instr i+3				IF	ID	EX	MEM	WB	
instr i+4					IF	ID	EX	MEM	WB

Instruction Types of MIPS

I - type instruction

Encodes: Loads and stores of bytes, words, half-words

All immediates (rd <- rs1 op immediate)

Conditional branch instructions (rs1 is register, rd unused)

Jump register, Jump and link register

(rd = 0, rs = destination, immediate = 0)

Instruction Types of MIPS
(Cont..)

R - type instruction

Register - register ALU operations: rd <- rs1 func rs2

Function encodes the data path operation: Add, Sub,...

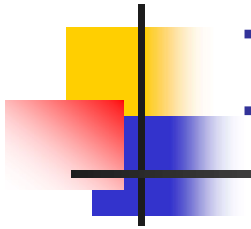
Read/Write special registers and moves

Instruction Types of MIPS
(Cont..)

J - type instruction

Jump and Jump and link

Trap and RFE



Instruction Fetch Cycle

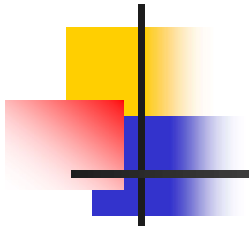
IR <- MEM[PC]

NPC <- PC +4

Instruction decode/register fetch (ID)

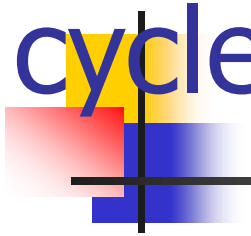
A <- Regs[IR6..10]
B <- Regs[IR11..15]
Imm <- ((IR16)16##IR16..31)

Execution/Effective address cycle (EX)



- **Memory reference:**
  ALUOutput <- A +Imm
- **Register-Register ALU instruction:**
  ALUOutput <- A *op* B
- **Register- Immediate ALU instruction:**
  ALUOutput <- A *op* Imm
- **Branch:**
  ALUOutput <- NPC + Imm
  Cond <- (A op 0)

Memory access/branch completion cycle (MEM)



Memory reference:

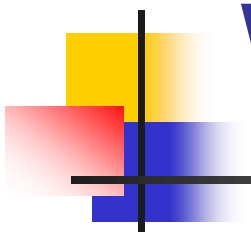
LMD <- Mem[ALUOutput] or Mem[ALUOutput] <- B

Branch:

if (cond) PC <- ALUOutput

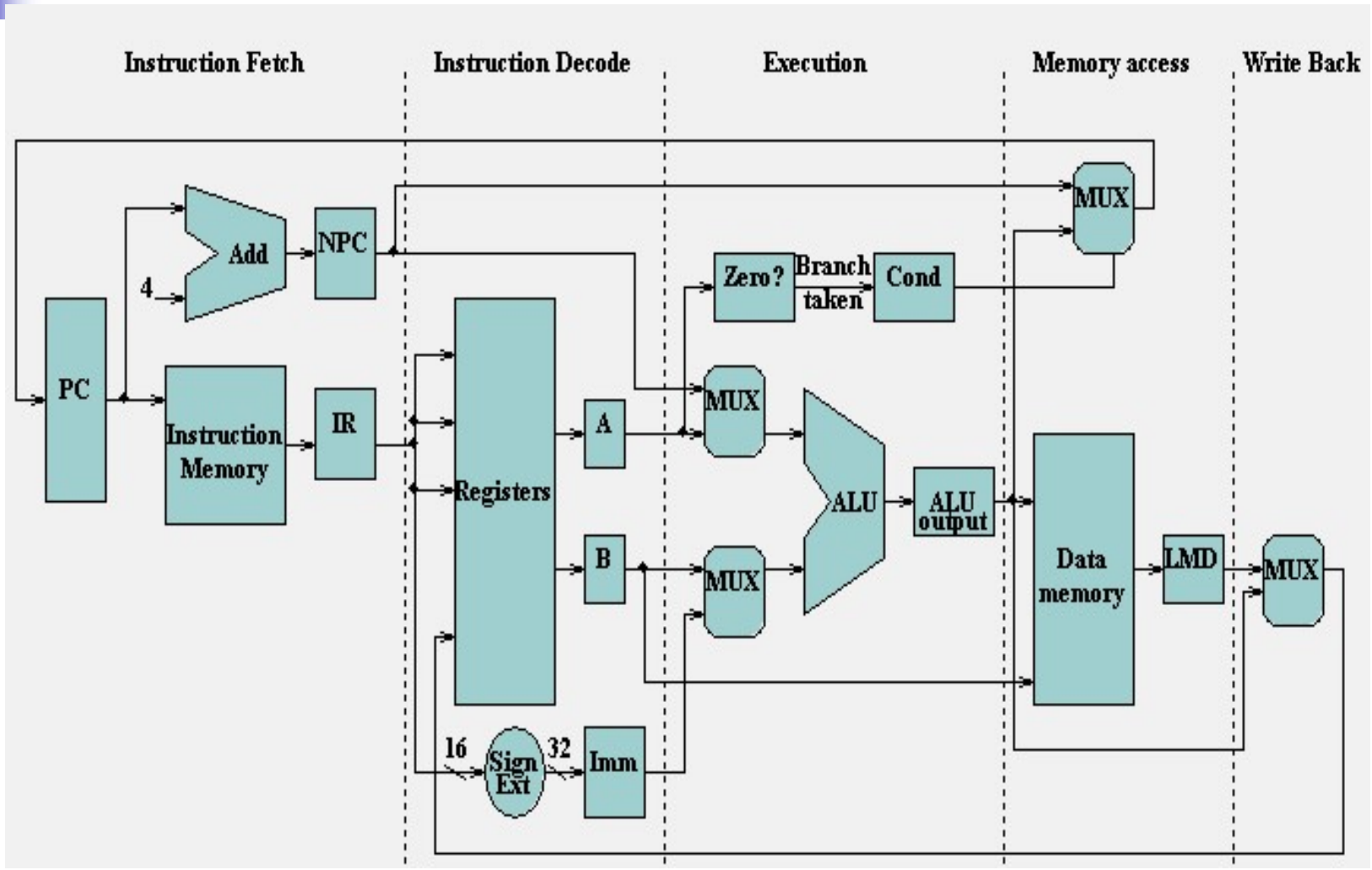
else PC <- NPC

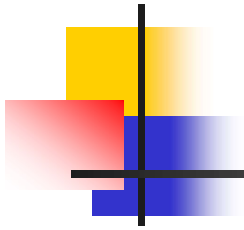
Write-back cycle (WB)



- Register-Register ALU instruction:
  Regs[IR16..20] <- ALUOutput Register
- Immediate ALU instruction:
  Regs[IR11..15] <- ALUOutput
- Load instruction:
  Regs[IR11..15] <- LMD

MIPS Pipeline processor

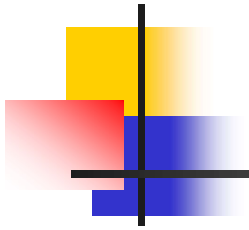




Pipeline Hazards

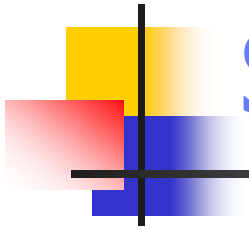
- There are situations, called **hazards**, that prevent the next instruction in the instruction stream from being executing during its designated clock cycle. Hazards reduce the performance from the ideal speedup gained by pipelining.

Types of hazards



There are three classes of hazards:

- Structural Hazards
- Data Hazards
- Control Hazards

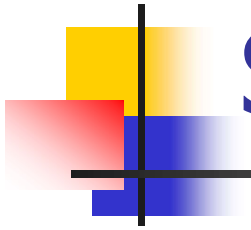


Structural Hazards

- Common instances of structural hazards arise when
- Some functional unit is not fully pipelined. Then a sequence of instructions using that unpipelined unit cannot proceed at the rate of one per clock cycle
  Some resource has not been duplicated enough to allow all combinations of instructions in the pipeline to execute.

Structural Hazards… Example

Clock cycle number								
Instr	1	2	3	4	5	6	7	8
Load	IF	ID	EX	MEM	WB			
Instr 1		IF	ID	EX	MEM	WB		
Instr 2			IF	ID	EX	MEM	WB	
Instr 3				IF	ID	EX	MEM	WB



Structural Hazards… Example

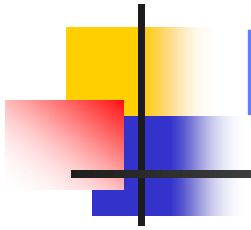
Clock cycle number									
Instr	1	2	3	4	5	6	7	8	9
Load	IF	ID	EX	**MEM**	WB				
Instr 1		IF	ID	EX	MEM	WB			
Instr 2			IF	ID	EX	MEM	WB		
Stall				bubble	bubble	bubble	bubble	bubble	
Instr 3					**IF**	ID	EX	MEM	WB

Structural Hazards… Example

Clock cycle number									
Instr	1	2	3	4	5	6	7	8	9
Load	IF	ID	EX	MEM	WB				
Instr 1		IF	ID	EX	MEM	WB			
Instr 2			IF	ID	EX	MEM	WB		
Instr 3				stall	IF	ID	EX	MEM	WB

Data hazards

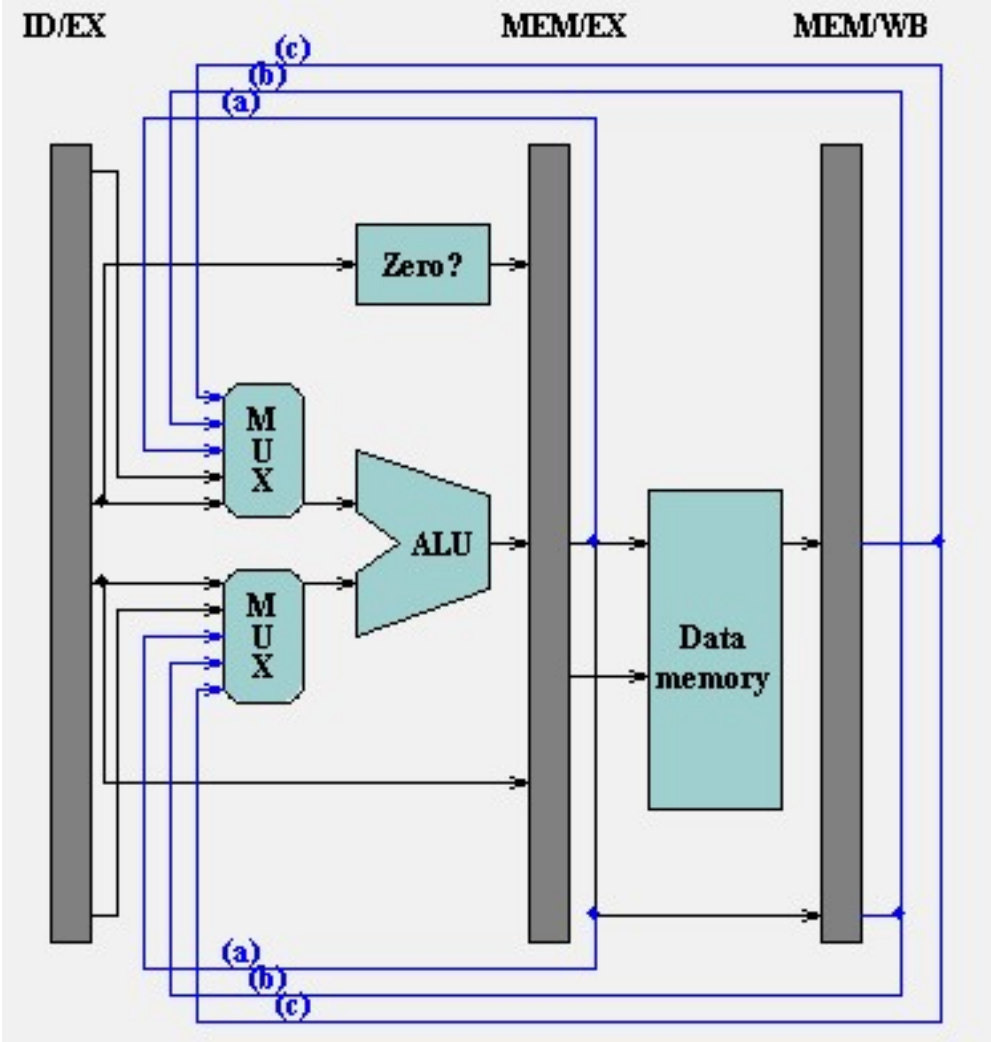
		1	2	3	4	5	6	7	8	9
ADD	R1, R2, R3	IF	ID	EX	MEM	WB				
SUB	R4, R5, R1		IF	ID_{sub}	EX	MEM	WB			
AND	R6, R1, R7			IF	ID_{and}	EX	MEM	WB		
OR	R8, R1, R9				IF	ID_{or}	EX	MEM	WB	
XOR	R10,R1,R11					IF	ID_{xor}	EX	MEM	WB

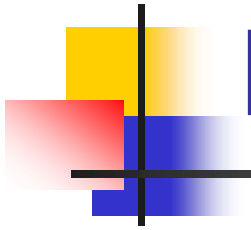


Data hazards Cont..

		1	2	3	4	5	6	7
ADD	**R1, R2, R3**	IF	ID	EX	MEM	**WB**		
SUB	R4, R5, **R1**		IF	**ID_{sub}**	EX	MEM	WB	
AND	R6, **R1, R7**			IF	**ID_{and}**	EX	MEM	WB

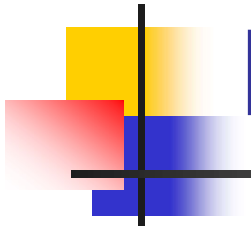
Data Forwarding





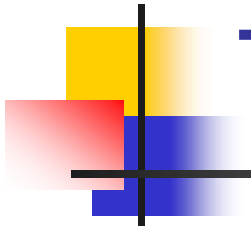
Data Forwarding Cont..

		1	2	3	4	5	6	7	8	9
ADD	R1, R2, R3	IF	ID	EX	MEM	WB				
SUB	R4, R5, R1		IF	stall	stall	ID_{sub}	EX	MEM	WB	
AND	R6, R1, R7			stall	stall	IF	ID_{and}	EX	MEM	WB



Data Forwarding Cont..

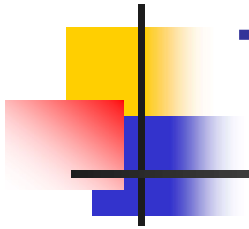
		1	2	3	4	5	6	7
ADD	R1, R2, R3	IF	ID	EX_{add}	MEM_{add}	WB		
SUB	R4, R5, R1		IF	ID	EX_{sub}	MEM	WB	
AND	R6, R1, R7			IF	ID	EX_{and}	MEM	WB



Types of Data Hazards

Consider two instructions i and j ,
i occuring before j

- RAR (read after read)
- RAW (read after write)
- WAW (write after write)
- WAR (write after read)

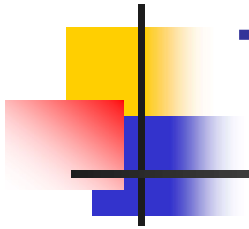


Types of Data Hazards Cont..

- RAR hazard

Ex: ADD r1,r2,r3

SUB r4,r5,r3

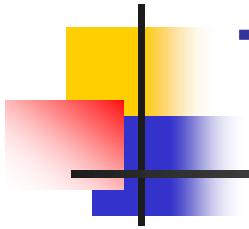


Types of Data Hazards Cont..

- RAW Hazard

Ex: ADD r1, r2, r3

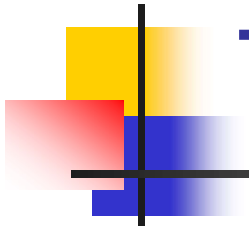
SUB r4, r5, r1



Types of Data Hazards Cont..

- WAW hazard

Ex: ADD r1, r2, r3
SUB r1, r5, r6



Types of Data Hazards Cont..

- WAR hazards

Ex: ADD r1, r2, r3

SUB r2, r5, r6

When Stalls are Required

		1	2	3	4	5	6	7	8
LW	R1, 0(R1)	IF	ID	EX	MEM	WB			
SUB	R4, R1, R5		IF	ID	EX_{sub}	MEM	WB		
AND	R6, R1 R7			IF	ID	EX_{and}	MEM	WB	
OR	R8, R1, R9				IF	ID	EX	MEM	WB

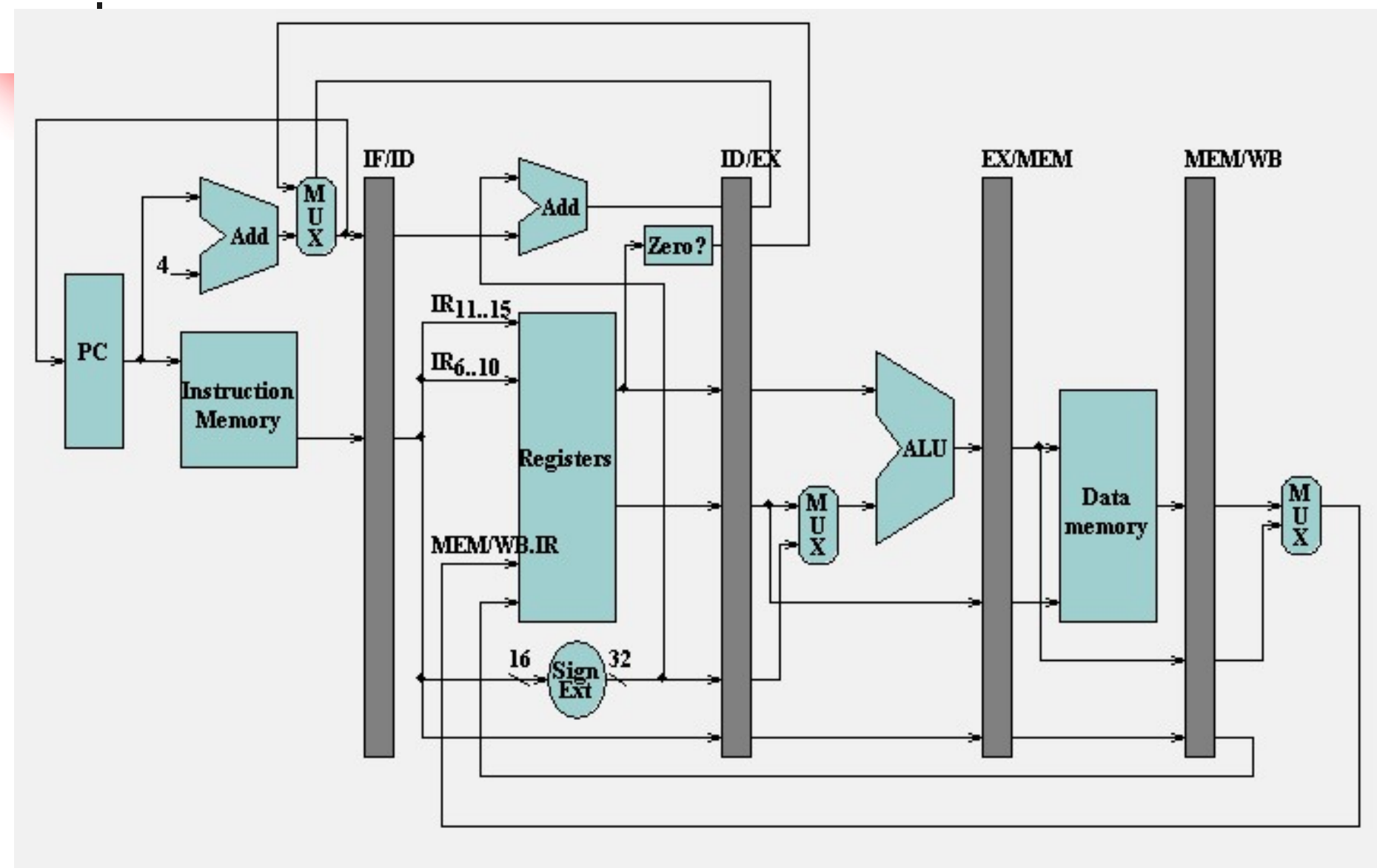
When Stalls are Required Cont..

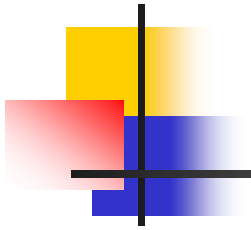
		1	2	3	4	5	6	7	8	9
LW	R1, 0(R1)	IF	ID	EX	MEM	WB				
SUB	R4, R1, R5		IF	ID	*stall*	EX_{sub}	MEM	WB		
AND	R6, R1 R7			IF	*stall*	ID	EX	MEM	WB	
OR	R8, R1, R9				*stall*	IF	ID	EX	MEM	WI

Control Hazards

Branch	IF	**ID**	EX	**MEM**	WB					
Branch successor		**IF(stall)**	***stall***	***stall***	IF	ID	EX	MEM	WB	
Branch successor +1						IF	ID	EX	MEM	WB

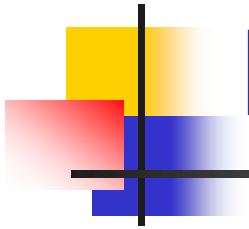
Control Hazards cont..





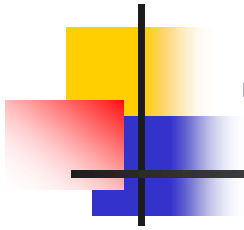
Control Hazards Cont..

Branch	IF	**ID**	EX	MEM	WB		
Branch successor		**IF(stall)**	IF	ID	EX	MEM	WB



Branch Prediction Schemes

- Stall pipeline
- Predict taken
- Predict not taken
- Delayed branch



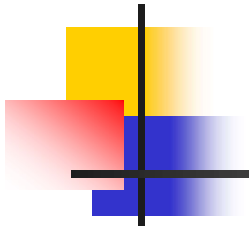
Stall the pipeline

The simplest scheme to handle branches is to freeze or flush the pipeline, holding or deleting any instructions after the branch until the branch destination is known. Advantage: simple both to software and hardware (solution described earlier)

Predict Not Taken

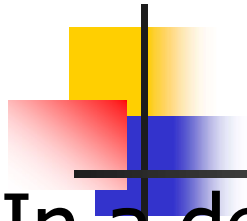
Untaken **Branch Instr**	IF	ID	EX	MEM	WB		
Instr i+1		IF	ID	EX	MEM	WB	
Instr i+2			IF	ID	EX	MEM	WB

Taken **Branch Instr**	IF	ID	EX	MEM	WB			
Instr i+1		IF	***idle***	***idle***	***idle***	***idle***		
Branch target			IF	ID	EX	MEM	WB	
Branch target+1				IF	ID	EX	MEM	WB



Predict Taken

An alternative scheme is to predict the branch as taken. As soon as the branch is decoded and the target address is computed, we assume the branch to be taken and begin fetching and executing at the target address.

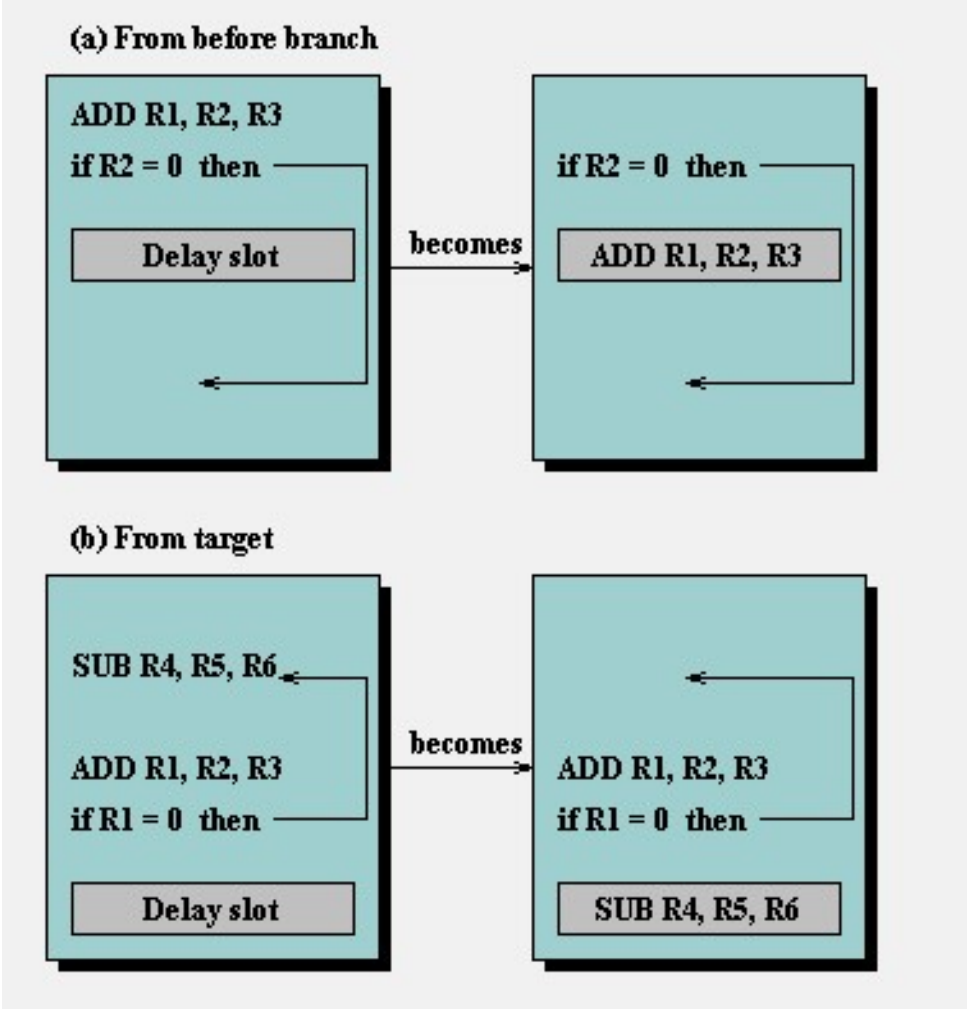


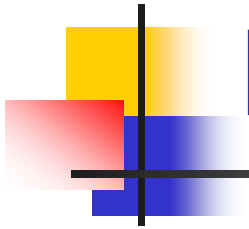
Delayed Branch

In a delayed branch, the execution cycle with a branch delay of length n is

Branch instr
sequential successor 1
sequential successor 2
. . . . .
sequential successor n
Branch target if taken

Delayed Branch Cont..





MIPS Pipeline Exceptions

- IF
  - Page Fault
  - Mis Aligned Memory Access
  - Memory protection violation
- ID
  - Invalid opcode

- EX :
- Overflow
- MEM
  - Page Fault
  - Mis Aligned Memory Access
  - Memory protection violation

- WB :
- None