Initial Design of Pipelined MIPS Processor

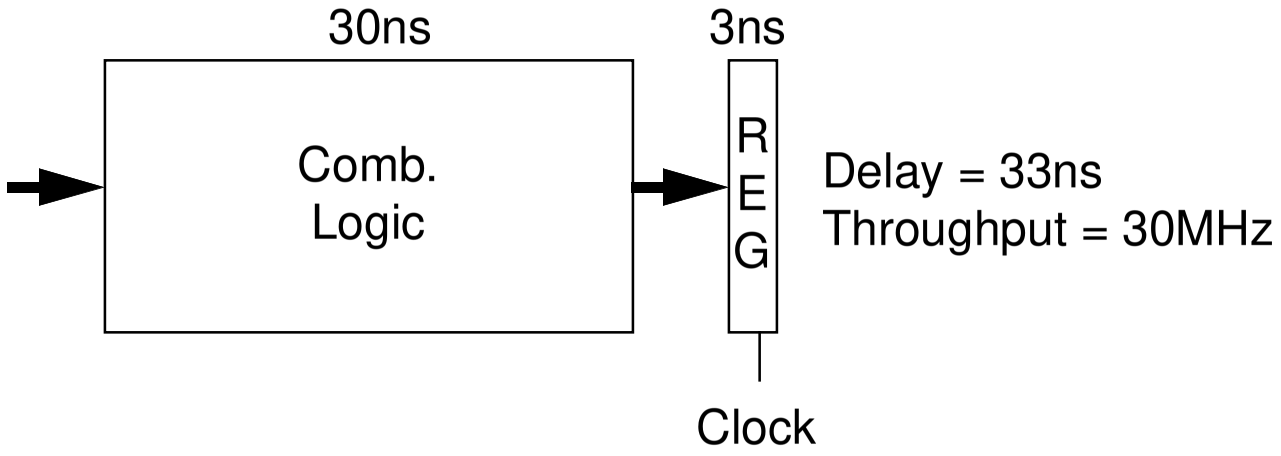
Feb 27, 1996

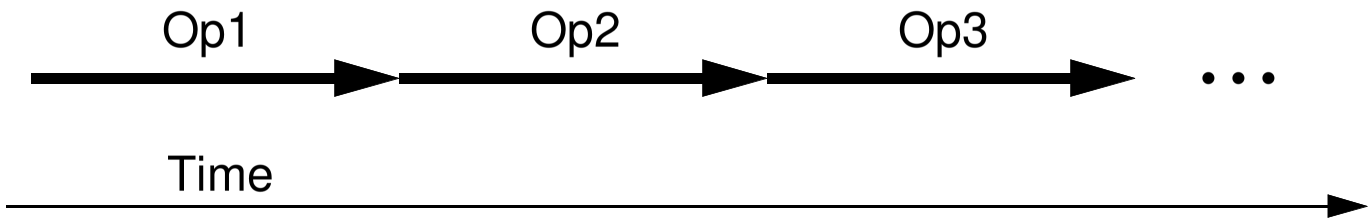
Topics

- General properties of pipelines
- Instruction format
- Single cycle design
- Inserting pipe registers
- Shortcomings

Illustration of Pipelining

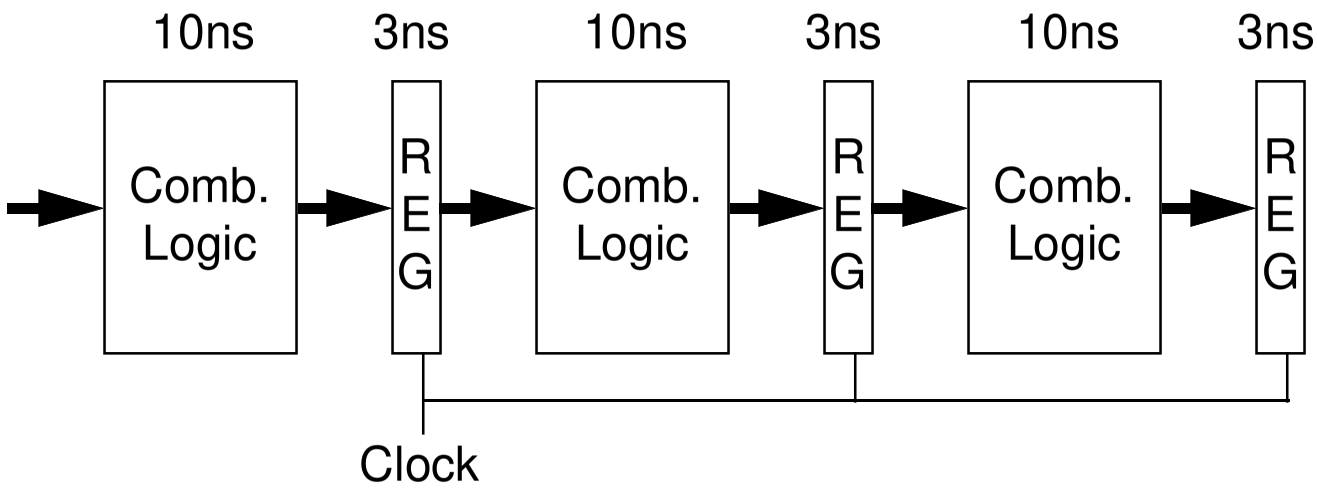
Unpipelined System

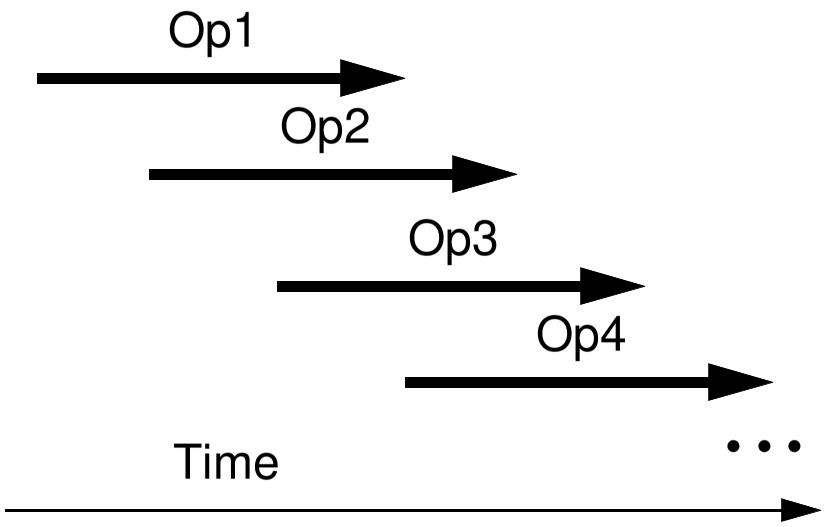




- **One operation must complete before next can begin**
- **Operations spaced 33ns apart**

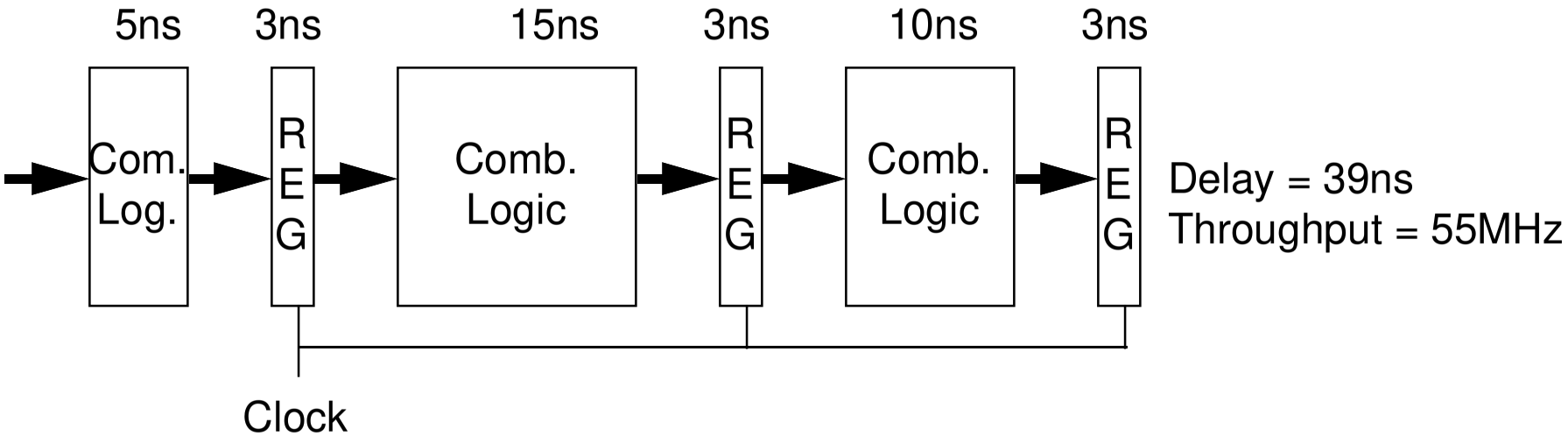
3 Stage Pipelining





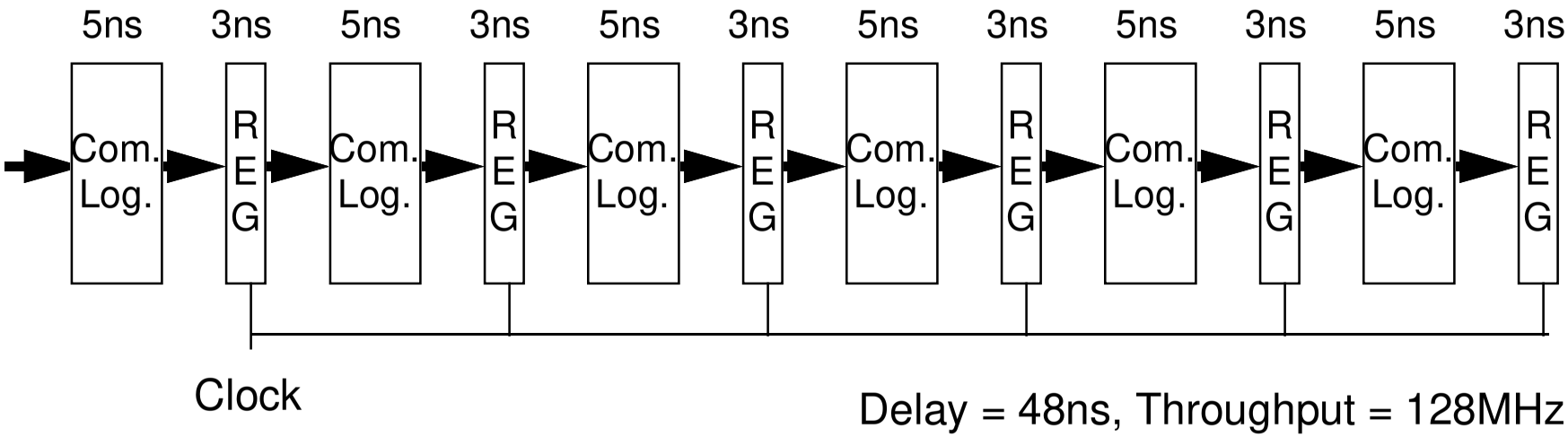
- **Space operations 13ns apart**
- **3 operations occur simultaneously**

Limitation: Nonuniform Pipelining



- **Throughput limited by slowest stage**
- **Must attempt to balance stages**

Limitation: Deep Pipelines



- **Diminishing returns as add more pipeline stages**
- **Register delays become limiting factor**
  - Increased latency
  - Small througput gains

Summary of Principles

Concept

- **Technique to increase throughput**
- **At some cost in latency**

Challenges

- **Fit all operations into uniform template**
- **Divide into equal length stages**
  - Throughput limited by slowest stage
  - Diminishing returns as increase number of stages

MIPS architecture subset

R-type instructions (add, sub, and, or, slt): rd <-- rs funct rt

0	rs	rt	rd	shamt	funct
31-26	25-21	20-16	15-11	10-6	5-0

RI-type instructions (addiu): rt <-- rs funct I16

0	rs	rt	I16
31-26	25-21	20-16	15-0

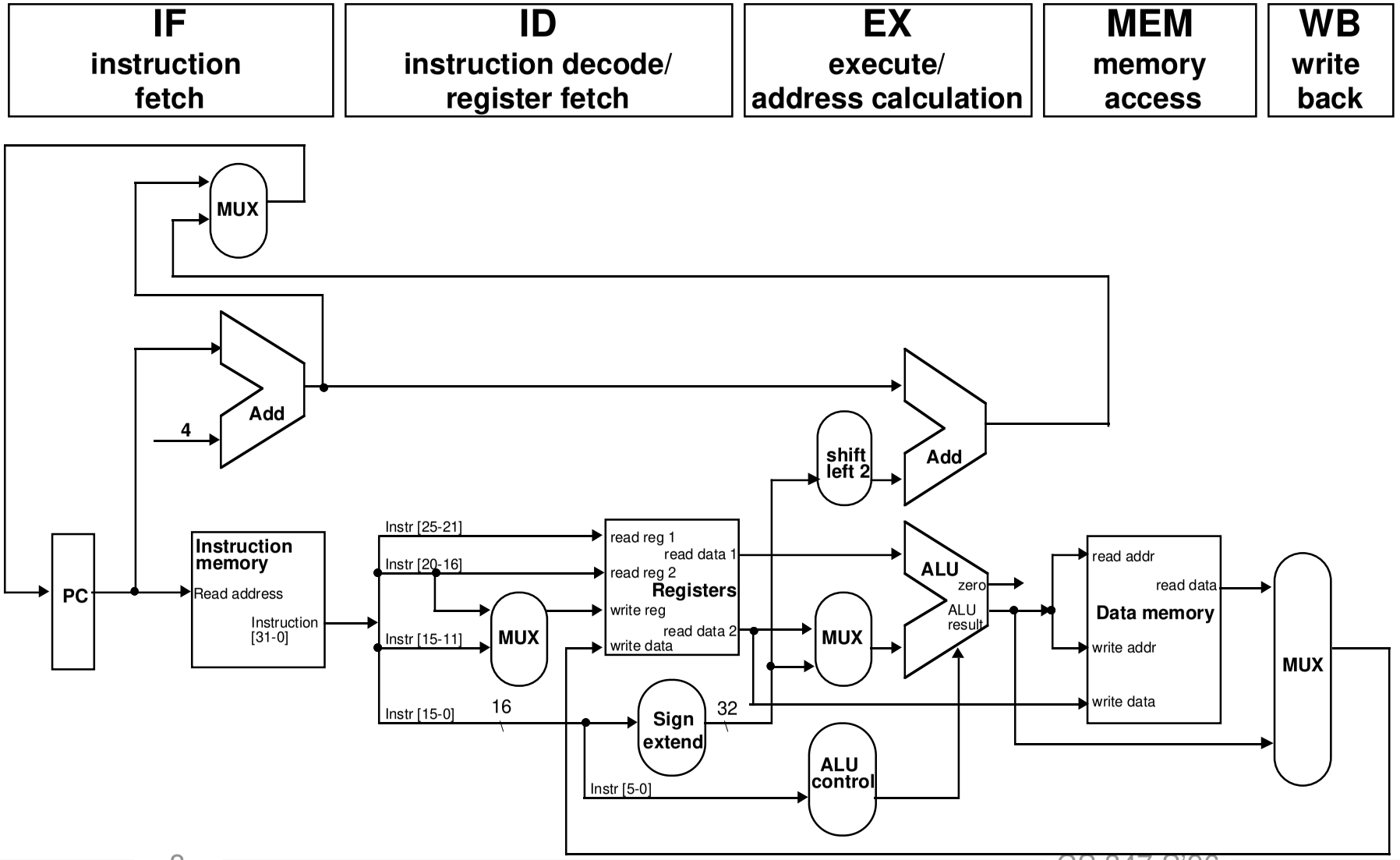
Load: rt <-- Mem[rs + I16] **Store:** Mem[rs + I16] <-- rt

35 or 43	rs	rt	I16
31-26	25-21	20-16	15-0

Branch equal: PC <-- (rs == rt) ? PC + 4 + I16 <<2 : PC + 4

4	rs	rt	I16
31-26	25-21	20-16	15-0

Datapath


IF
instruction fetch
ID
instruction decode/ register fetch
EX
execute/ address calculation
MEM
memory access
WB
write back
MUX
Add
4
PC
Instruction memory
Read address
Instruction [31-0]
Instr [25-21]
Instr [20-16]
Instr [15-11]
Instr [15-0]
Instr [5-0]
read reg 1
read reg 2
Registers
write reg
write data
read data 1
read data 2
16
Sign extend
32
shift left 2
Add
ALU
zero
ALU result
read addr
read data
Data memory
write addr
write data
ALU control
MUX

R-type instructions

IF: Instruction fetch

- IR <-- IMemory[PC]
- PC <-- PC + 4

ID: Instruction decode/register fetch

- A <-- Register[IR[25:21]]
- B <-- Register[IR[20:16]]

Ex: Execute

- ALUOutput <-- A op B

MEM: Memory

- nop

WB: Write back

- Register[IR[15:11]] <-- ALUOutput

RI-type instructions

IF: Instruction fetch

- IR <-- IMemory[PC]
- PC <-- PC + 4

ID: Instruction decode/register fetch

- A <-- Register[IR[25:21]]
- B <-- Register[IR[20:16]]

Ex: Execute

- ALUOutput <-- A op SignExtend(IR[15:0])

MEM: Memory

- nop

WB: Write back

- Register[IR[20:16]] <-- ALUOutput

Load instruction

IF: Instruction fetch

- IR <-- IMemory[PC]
- PC <-- PC + 4

ID: Instruction decode/register fetch

- A <-- Register[IR[25:21]]
- B <-- Register[IR[20:16]]

Ex: Execute

- ALUOutput <-- A + SignExtend(IR[15:0])

MEM: Memory

- Mem-Data <-- DMemory[ALUOutput]

WB: Write back

- Register[IR[20:16]] <-- Mem-Data

Store instruction

IF: Instruction fetch

- IR <-- IMemory[PC]
- PC <-- PC + 4

ID: Instruction decode/register fetch

- A <-- Register[IR[25:21]]
- B <-- Register[IR[20:16]]

Ex: Execute

- ALUOutput <-- A + SignExtend(IR[15:0])

MEM: Memory

- DMemory[ALUOutput] <-- B

WB: Write back

- nop

Branch on equal

IF: Instruction fetch

- IR <-- IMemory[PC]
- PC <-- PC + 4

ID: Instruction decode/register fetch

- A <-- Register[IR[25:21]]
- B <-- Register[IR[20:16]]

Ex: Execute

- Target <-- PC + SignExtend(IR[15:0]) << 2
- Z <-- (A - B == 0)

MEM: Memory

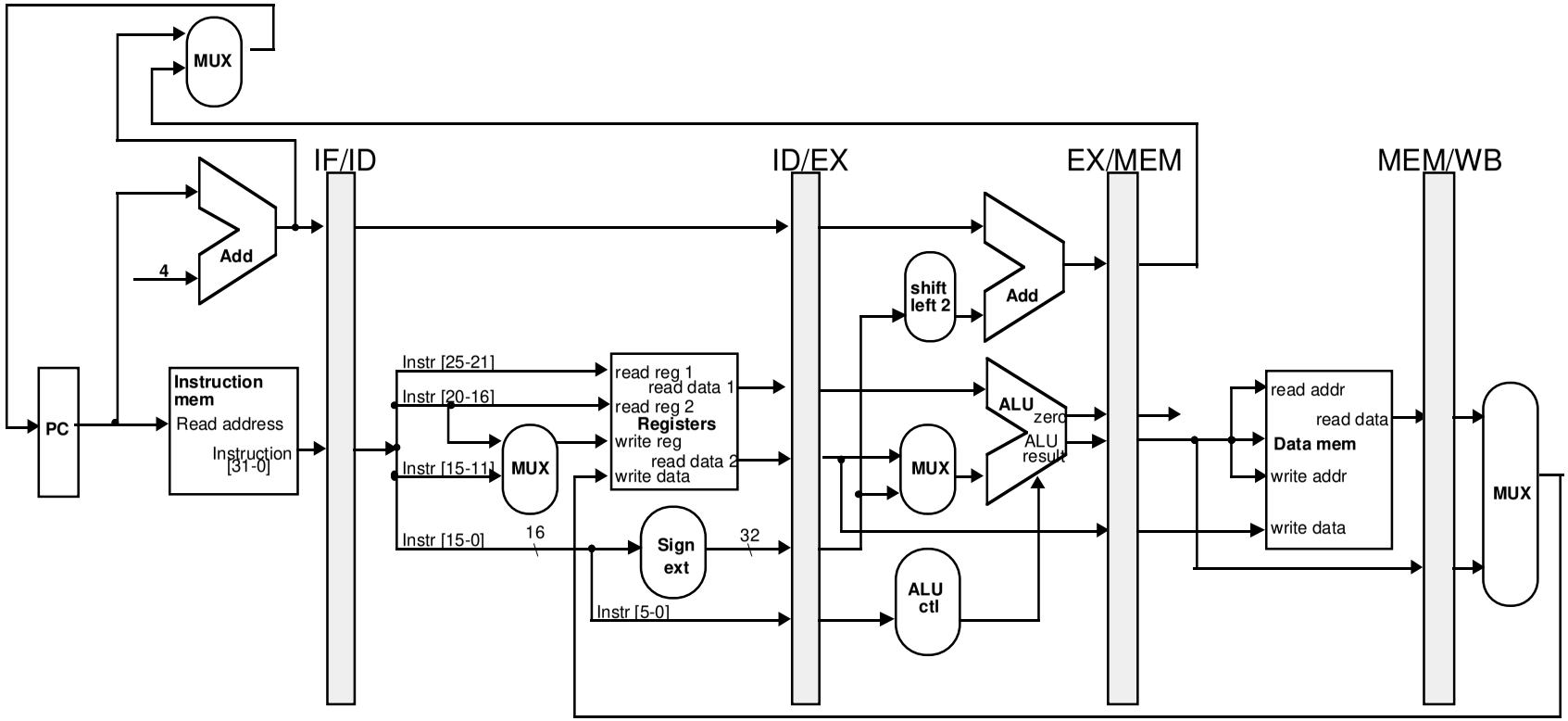
- If (Z) PC <-- target

WB: Write back

- nop

Is this a delayed branch?

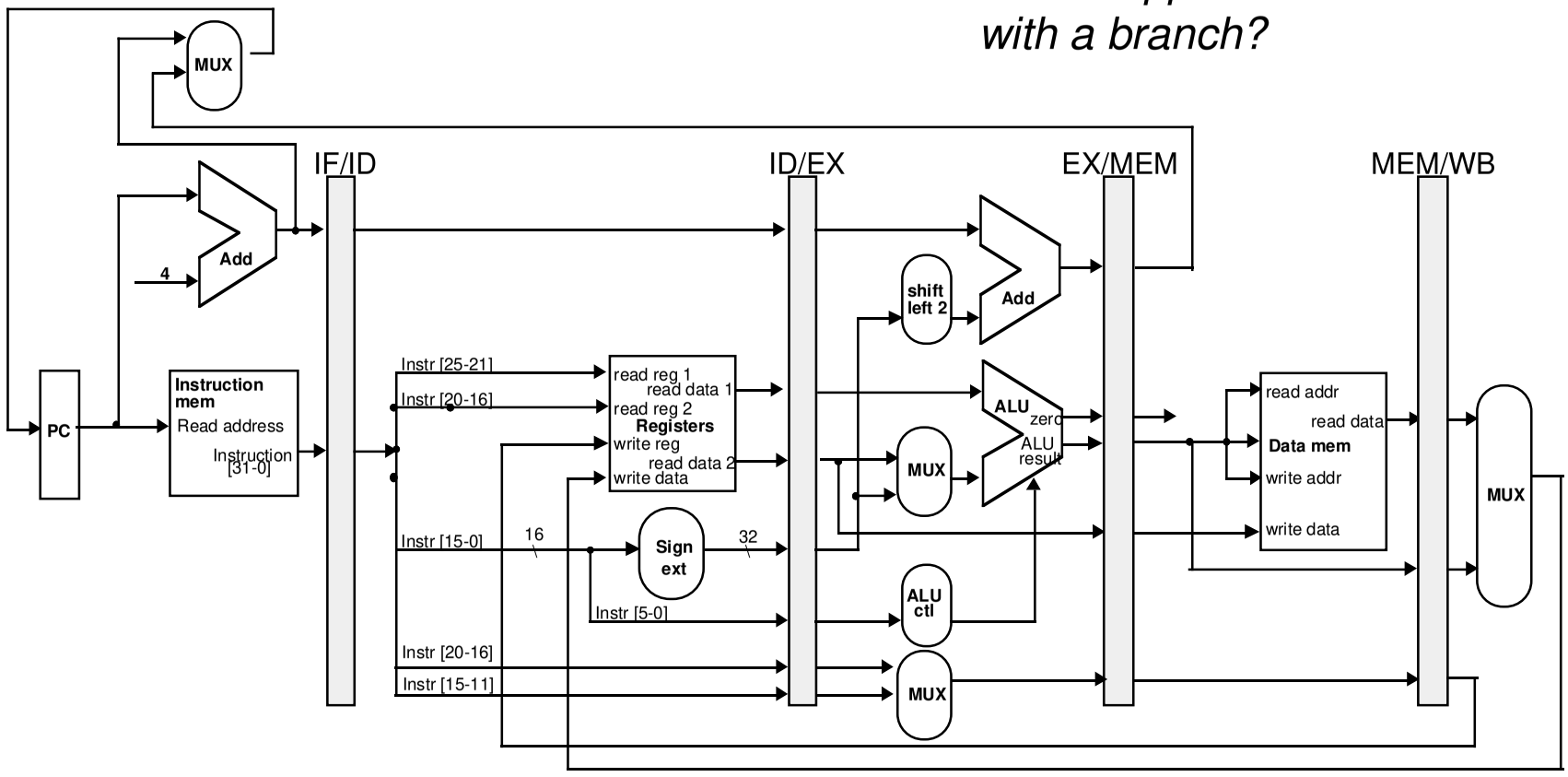
Adding pipe registers



Where would this write back?

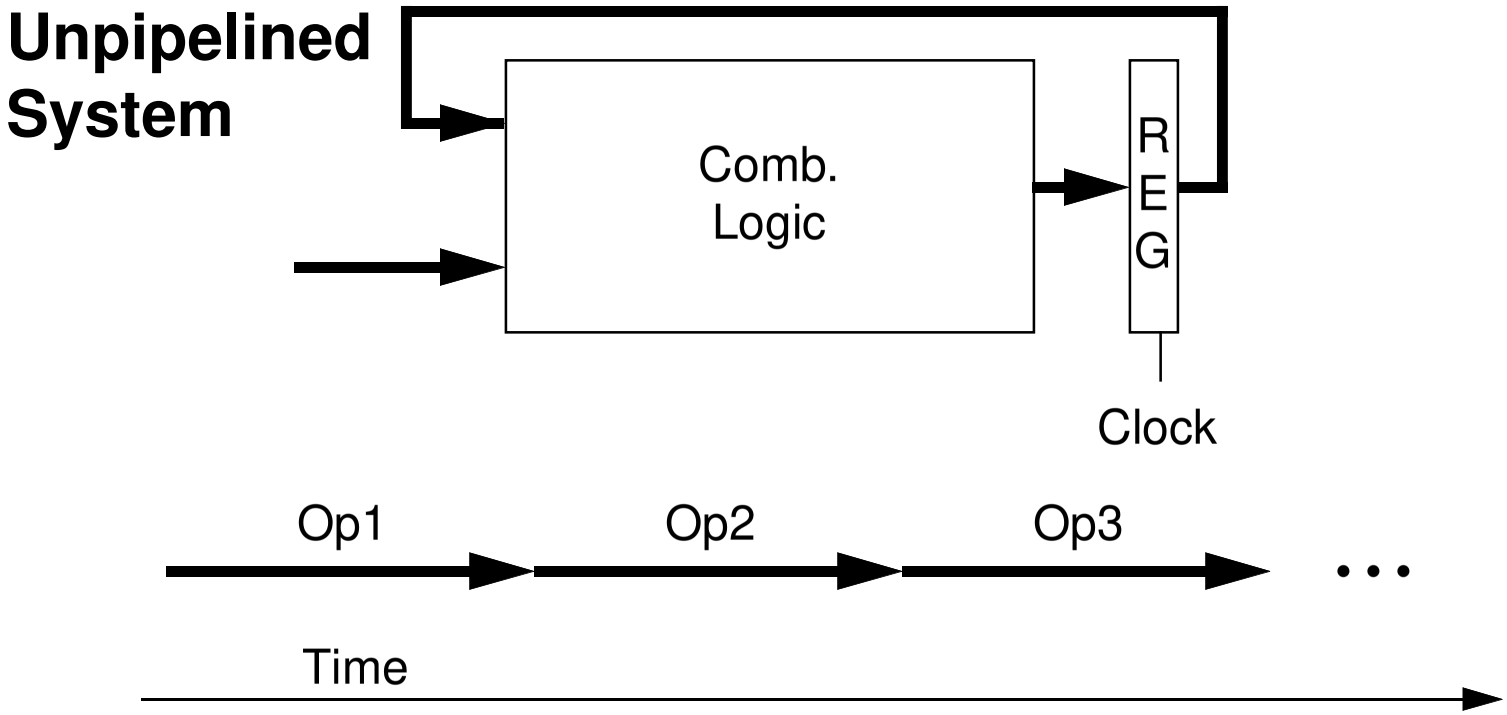
Pipelined datapath

What happens with a branch?



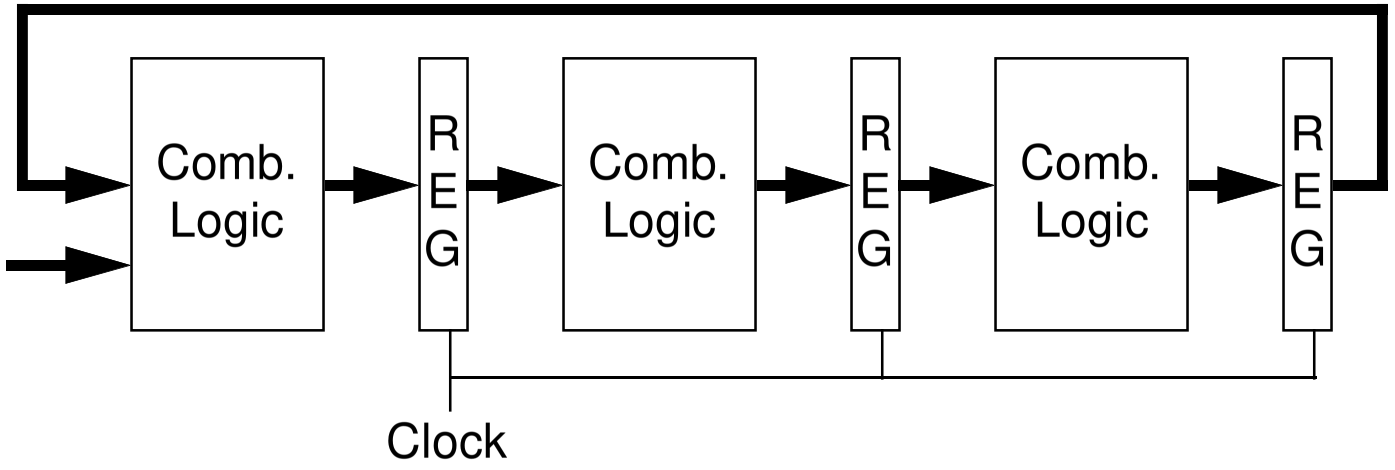
How about that writeback loop?

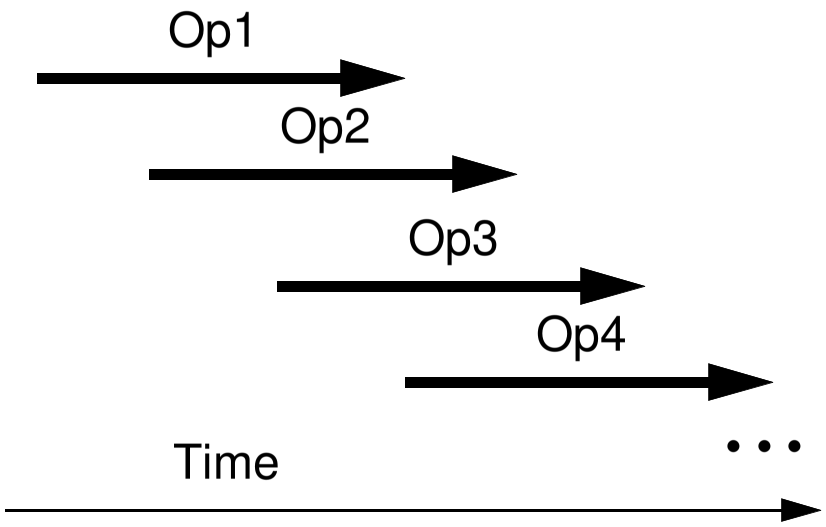
Pipelining Sequential Ops.



- **Each operation gets result from previous operation**

3 Stage Sequential Pipeline



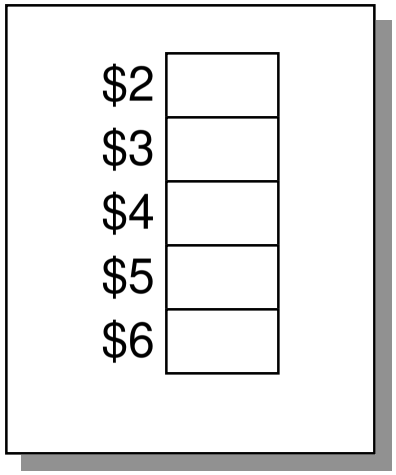


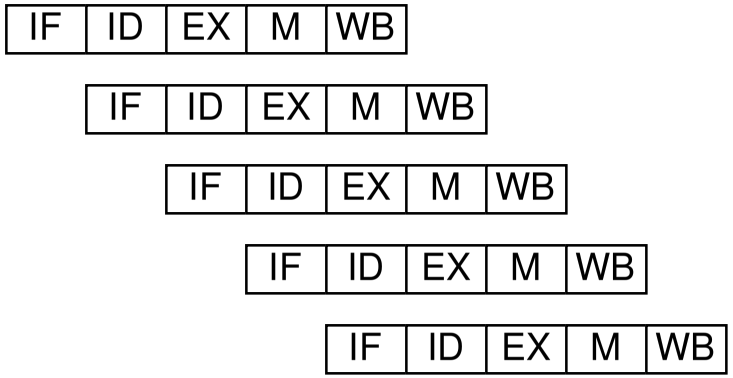
- **Op4 gets result from Op1 !**
- ***Pipeline Hazard***

Data Hazards in MIPS Pipeline

Problem

- **Registers read in ID, and written in WB**
- **Must resolve conflict between instructions competing for register array**
  - Generally do write back in first half of cycle, read in second
- **But what about intervening instructions?**
- **E.g., suppose initially $2 is zero:**



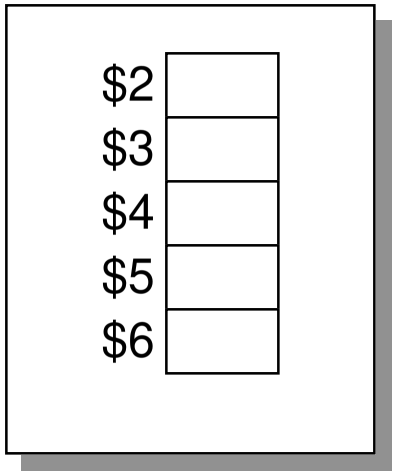


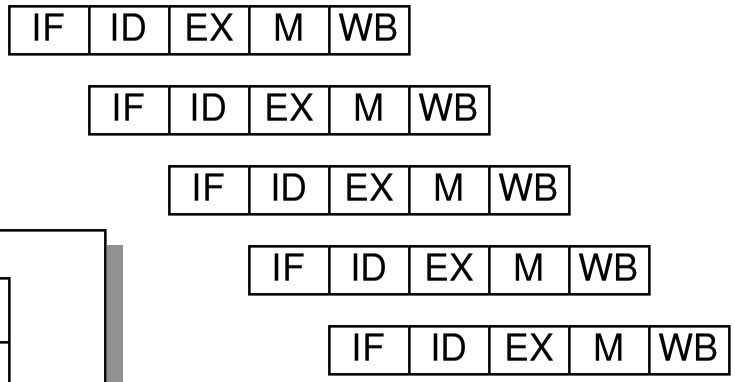

```
addiu $2, $0, 63
addiu $3, $2, 0
addiu $4, $2, 0
addiu $5, $2, 0
addiu $6, $2, 0
```

Control Hazards in MIPS Pipeline

Problem

- **Instruction fetched in IF, branch condition set in MEM**
- **When does branch take effect?**
- **E.g.: assume initially that all registers = 0**





beq $0, $0, target

addiu $2, $0, 63

addiu $3, $0, 63

addiu $4, $0, 63

addiu $5, $0, 63

target: addiu $6, $0, 63