Pipelining

- doesn't help latency of single task, helps the throughput of **entire** workload
- Multiple tasks operating simultaneously using **different resources**

Five Execution Stages

IF: Instruction Fetch

- Read instruction from memory using the address in PC and put it in **IF/ID** register
- PC address is incremented by **4** and then written back to the PC for next instruction

ID: Instruction Decode and Register Read

IF/ID supplies:	**ID/EX receives:**
• **Register numbers** for reading two registers • 16-bit **offset** to be sign-extended to 32-bit • PC + 4	• Data values read from register file • 32-bit immediate value • PC + 4

EX: Execute an operation or calculate an address

ID/EX supplies:	**EX/MEM receives:**
• Data values read from register file • 32-bit immediate value • **PC + 4**	• (PC + 4) + (Imm x 4) • ALU result • **isZero?** signal • Data Read 2 from RegFile

MEM: Access an operand in data memory

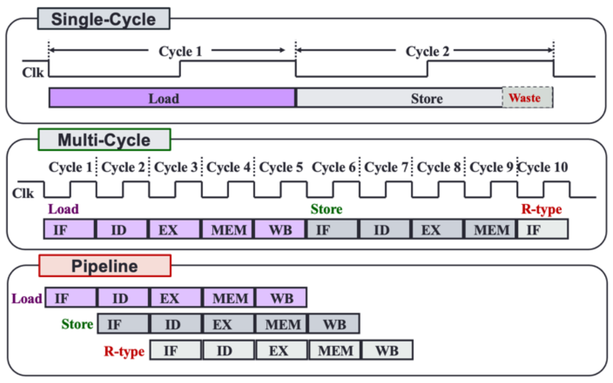
EX/MEM supplies:	**MEM/WB receives:**
(PC + 4) + (Imm x 4) ALU result **isZero?** signal Data Read 2 from RegFile	• ALU result • Memory read data

WB: Write back the result into a register

MEM/WB supplies	**At the end of a cycle**
ALU result Memory read data	Result is written back to register file (if applicable) using WR number*

*Pass "Write register" number from ID/EX through EX/MEM to MEM/WB pipeline register for use in WB stage

Instructions Execution Time



Single cycle:

- $CT_{seq} = \max\left(\sum_1^k T_k\right)$
- $\text{Time}_{seq} = I \times CT_{seq}$ for *I* instructions

Multiple cycle:

- $CT_{\text{multi}} = \max\left(T_k\right)$
- $\text{Time}_{\text{multi}} = I \times CPI \times CT_{\text{multi}}$

  CPI = num of cycles per instruction

Inst	IF	ID	EX	MEM	WB	Total
branch	X	X	X			3
R-type	X	X	X		X	4
I-type	X	X	X		X	4
lw	X	X	X	X	X	5
sw	X	X	X	X		4
j	X	X				2

* Assume that the j instruction only uses the fetch and decode stages

Pipeline cycle:

- $CT_{\text{pipeline}} = \max\left(T_k\right) + T_d$

  T_d=Overhead for pipelining, e.g. pipeline register
- Cycles needed for **I** instructions: $I + N - 1$

  (need $N -$ 1 cycles to fill up the pipeline)
- Execution Time for **I** instructions:

$$\text{Time}_{pipeline} = (I + N - 1) \times CT_{pipeline}$$

Assumptions for **ideal case**:

- Every stage takes the same amount of time

$$\sum_{k=1}^{N} T_k = N \times T_1 (1)$$

- No pipeline overhead: $T_d = 0$
- $I >> N$ (Number of instructions is much larger than number of stages)

$$Speedup_{pipeline} = \frac{Time_{seq}}{Time_{pipeline}} = \frac{I \times \sum_{k=1}^{N} T_k}{(I + N - 1) \times (\max(T_k) + T_d)} = \frac{I \times N \times T_1}{(I + N - 1) \times T_1}$$

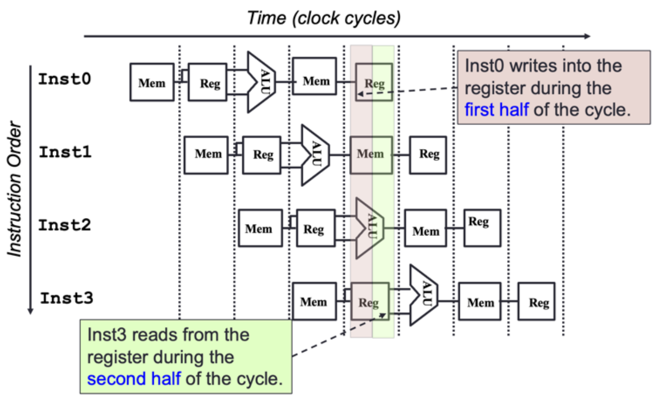
$$\approx \frac{I \times N \times T_1}{I \times T_1} = N$$

Pipeline Hazards: Problems that prevent next instruction from immediately following previous instruction

Structural Hazards

Recall that registers are very **fast memory**.

Solution: Split cycle into half; first half for writing into a register; second half for reading from a register.


Time (clock cycles)
Instruction Order
Inst0
Inst1
Inst2
Inst3
Mem
Reg
ALU
Inst0 writes into the register during the first half of the cycle.
Inst3 reads from the register during the second half of the cycle.

Data Hazards

Register contention is the cause of **dependency**

(1) **R**ead-**A**fter-**W**rite **(true data dependency):** Occurs when a later instruction **reads** from the dest. register **written** by an earlier instruction

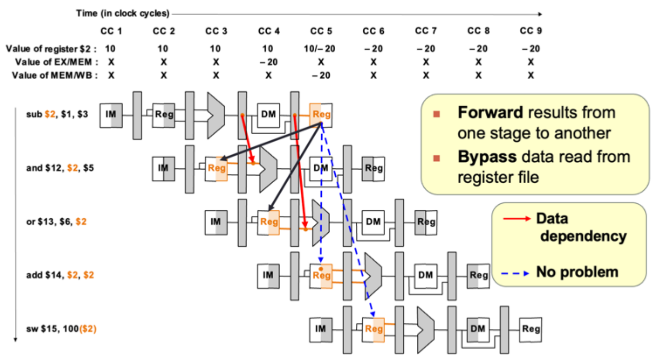
(2) **WAR**: **W**rite-**a**fter-**R**ead dependency

(3) **WAW**: **W**rite-**a**fter-**W**rite dependency

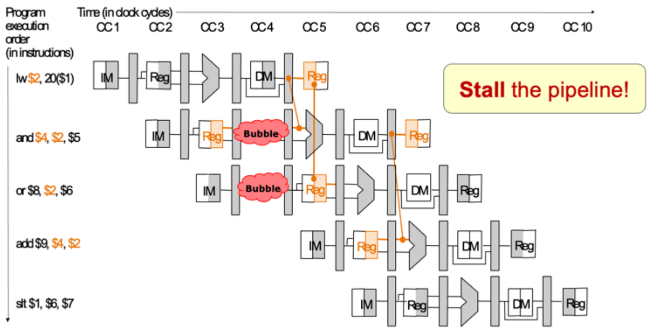
Fortunately, (2), (3) **do not** cause any pipeline hazards **unless** when instructions are executed out of program order, i.e. in Modern SuperScalar Processor

Forward

(1) No delay


Time (in clock cycles)
CC 1 CC 2 CC 3 CC 4 CC 5 CC 6 CC 7 CC 8 CC 9
Value of register $2 : 10 10 10 10 10/–20 –20 –20 –20 –20
Value of EX/MEM : X X X –20 X X X X X
Value of MEM/WB : X X X X –20 X X X X
sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
IM
Reg
DM
Forward results from one stage to another
Bypass data read from register file
Data dependency
No problem

(2) for **lw** , data is only ready after MEM stage may incur extra 1 cycle delay!



Control Hazards (Instruction Dependency)

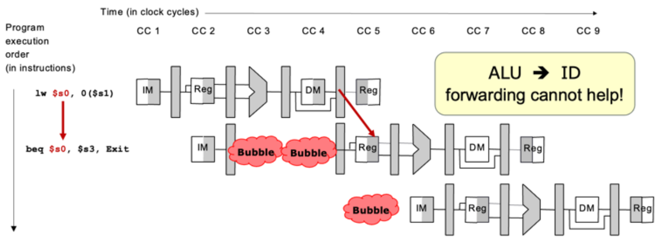
Control flow is the cause of dependency

Early Branch Resolution:

- Make decision in **ID** stage instead of **MEM**
- Move branch target address calculation
- Move register comparison→not use ALU for register comparison any more (can be done with a few gates)

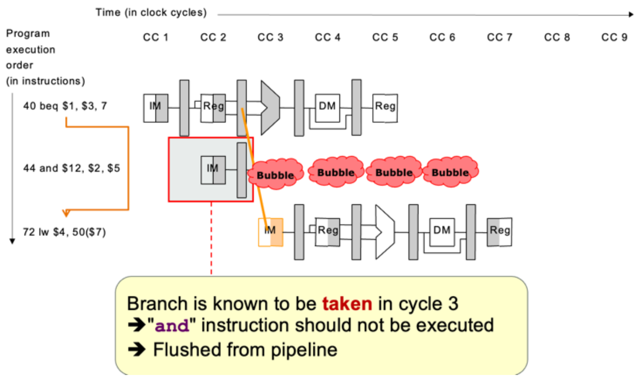
(1) if the register(s) involved in the comparison is produced by **preceding** instruction: need further stall

<u>ended up with 3 total stall cycles</u>


Time (in clock cycles)
CC 1 CC 2 CC 3 CC 4 CC 5 CC 6 CC 7 CC 8 CC 9
Program execution order (in instructions)
lw $s0, 0($s1)
beq $s0, $s3, Exit
IM
Reg
DM
Bubble
ALU → ID forwarding cannot help!

Branch Prediction: Guess outcome before it is produced

- All branches are assumed to be **not taken**
- Fetch the successor instruction and start pumping it through the pipeline stages
- When the actual branch outcome is known:
  Not taken→No pipeline stall
  Taken: Wrong instructions in the pipeline→**Flush** successor instruction from the pipeline


Time (in clock cycles)
CC 1 CC 2 CC 3 CC 4 CC 5 CC 6 CC 7 CC 8 CC 9
Program execution order (in instructions)
40 beq $1, $3, 7
44 and $12, $2, $5
72 lw $4, 50($7)
IM
Reg
DM
Bubble
Branch is known to be taken in cycle 3
→"and" instruction should not be executed
→ Flushed from pipeline

MISC (From PYP)

- Predict not taken is easier to implement, as PC+4 is already computed.
- To implement predict taken, we need to compute the target branch address quickly **in the first cycle**. That requires additional hardware.

Delayed Branching:

- if the branch outcome takes X number of cycles to be known, move non-control dependent instructions into the X slots following a branch
- aka branch delay slot (the 'no-op' slots)
- instructions that will be executed regardless of the branch outcome
- In MIPS processor, Branch-Delay slot = **1** (with the early branch)