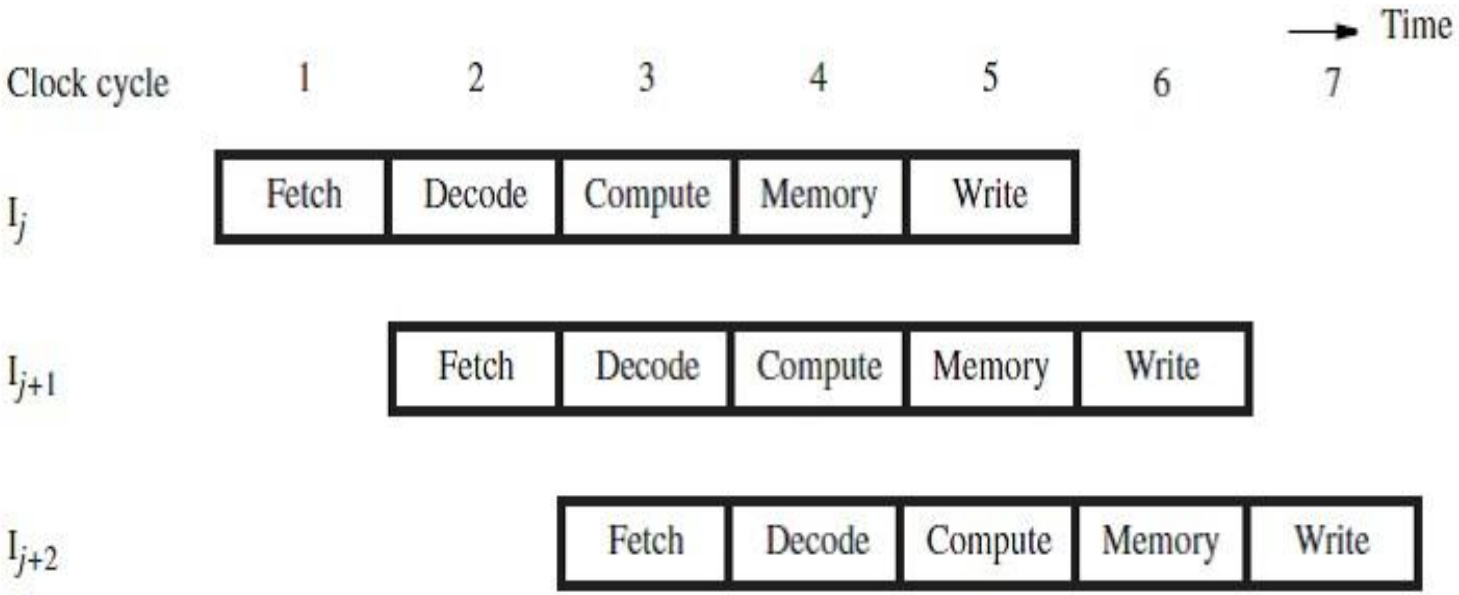
Computer Organization & Architecture

Pipelining

- Speed of execution of programs is influenced by many factors.
- One way to improve performance is to use faster circuit technology to implement the processor and the main memory.
- Another possibility is to arrange the hardware so that more than one operation can be performed at the same time.
  - Number of operations performed per second is increased, even though time needed to perform any one operation is not changed.

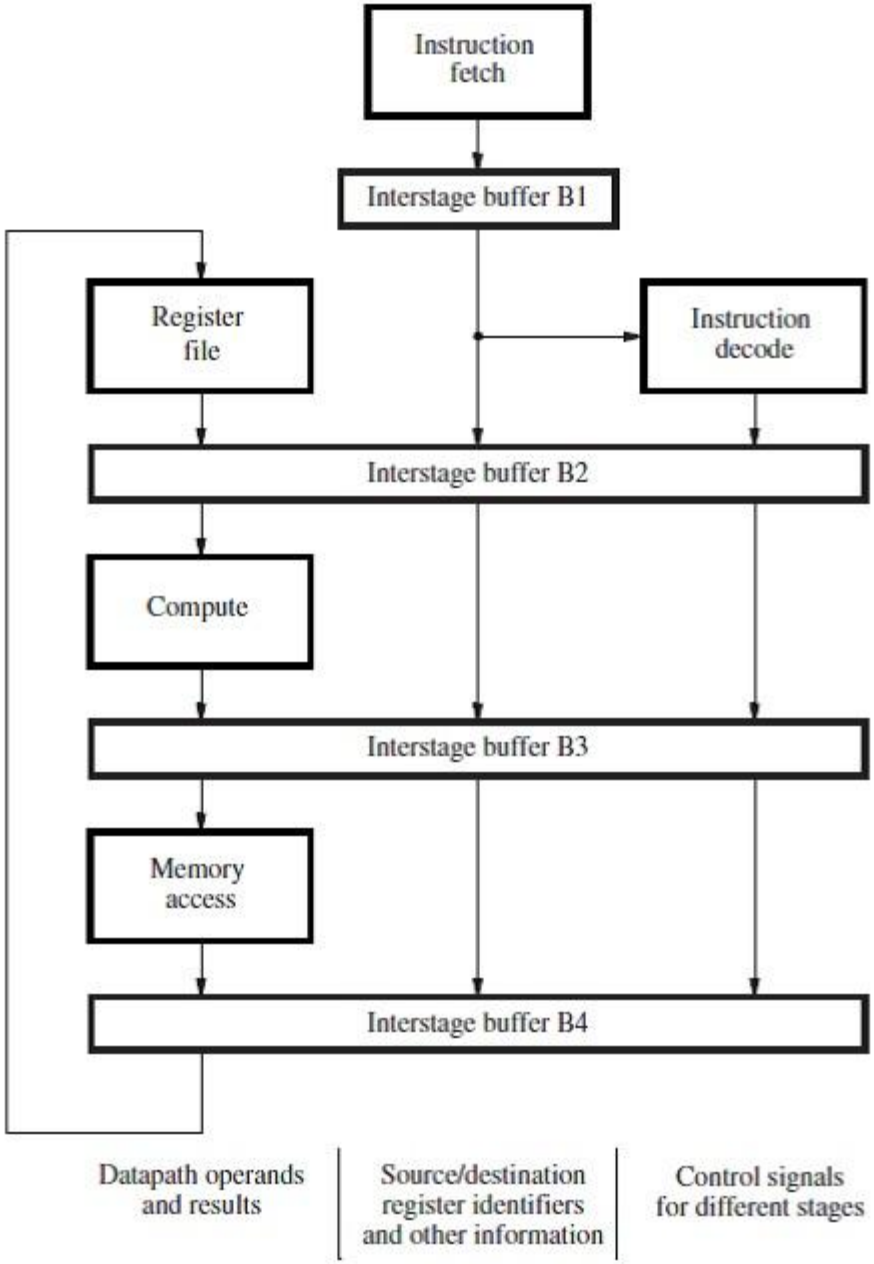
Pipelining

- Pipelining is a particularly effective way of organizing concurrent activity in a computer system.
  - Frequently encountered in manufacturing plants, where pipelining is commonly known as an assembly-line operation
- Consider how the idea of pipelining can be used in a computer.
  - The five-stage processor organization and the corresponding datapath allow instructions to be fetched and executed one at a time.
  - It takes five clock cycles to complete the execution of each instruction.
  - Rather than wait until each instruction is completed, instructions can be fetched and executed in a pipelined manner


Time
Clock cycle
1
2
3
4
5
6
7
I_j
Fetch
Decode
Compute
Memory
Write
I_{j+1}
Fetch
Decode
Compute
Memory
Write
I_{j+2}
Fetch
Decode
Compute
Memory
Write

Pipeline organization

- At any given time, each stage of the pipeline is processing a different instruction.
- Information such as register addresses, immediate data, and the operations to be performed must be carried through the pipeline as each instruction proceeds from one stage to the next.
  - This information is held in interstage buffers.
  - These include registers RA, RB, RM, RY, and RZ, the IR and PC-Temp registers


Instruction fetch
Interstage buffer B1
Register file
Instruction decode
Interstage buffer B2
Compute
Interstage buffer B3
Memory access
Interstage buffer B4
Datapath operands and results
Source/destination register identifiers and other information
Control signals for different stages

Pipelining issues

- There are times when it is not possible to have a new instruction enter the pipeline in every cycle.
- Consider case of two instructions, I_j and I_{j+1}, where the destination register for instruction I_j is a source register for instruction I_{j+1}.
  - The result of instruction I_j is not written into the register file until cycle 5, but it is needed earlier in cycle 3 when the source operand is read for instruction I_{j+1}.
  - If execution proceeds the result of instruction I_{j+1} would be incorrect because the arithmetic operation would be performed using the old value of the register in question.
  - To obtain the correct result, it is necessary to wait until the new value is written into the register by instruction I_j
  - Hence, instruction I_{j+1} cannot read its operand until cycle 6, which means it must be stalled in the Decode stage for three cycles.

Pipelining issues

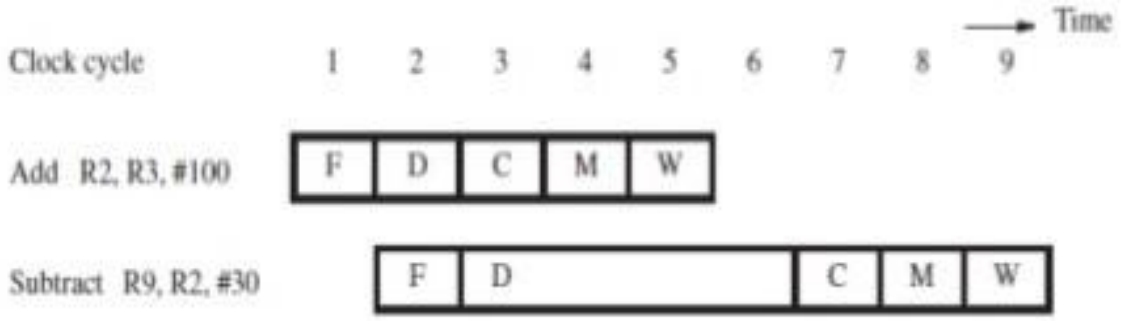
- While instruction I_{j+1} is stalled, instruction I_{j+2} and all subsequent instructions are similarly delayed.
  - New instructions cannot enter the pipeline, and the total execution time is increased.
  - Any condition that causes the pipeline to stall is called a **hazard**.
  - Above is an example of a data hazard, where the value of a source operand of an instruction is not available when needed.
  - Other hazards arise from memory delays, branch instructions, and resource limitations.

Data dependencies

Add R2, R3, #100

Subtract R9, R2, #30

- The destination register R2 for the Add instruction is a source register for the Subtract instruction.
- There is a data dependency between these two instructions, because register R2 carries data from the first instruction to the second.



Data dependencies

- The control circuit must first recognize the data dependency when it decodes the Subtract instruction in cycle3
  - By comparing its source register identifier from interstage buffer B1 with the destination register identifier of the Add instruction that is held in interstage buffer B2.
- Then, the Subtract instruction must be held in interstage buffer B1 during cycles 3 to5.
  - Meanwhile, the Add instruction proceeds through the remaining pipeline stages.
- In cycles 3 to 5, as the Add instruction moves ahead, control signals can be set in interstage buffer B2 for an implicit NOP (No-operation) instruction that does not modify the memory or the register file.
- Each NOP creates one clock cycle of idle time, called a bubble, as it passes through the Compute, Memory, and Write stages to the end of the pipeline.

Data Hazards: RAW

- A read after write (RAW) data hazard refers to a situation where an instruction refers to a result that has not yet been calculated or retrieved.
- This can occur because even though an instruction is executed after a prior instruction, the prior instruction has been processed only partly through the pipeline.

- Ex. I_j: ADD R2, R3, R5
  $I_{j+1:}$ ADD R4, R3, R2

  - The first instruction is calculating a value to be saved (write) in register R2
  - The second is going to use this value (read) to compute a result for register R4.
  - Operands are fetched for the successive operation and the results from the first have not yet been saved, and hence a RAW data dependency occurs.
  - Operand forwarding can help

Data Hazards: WAR

- A write after read (WAR) data hazard represents a problem with concurrent execution
- Ex. I_j: ADD R4, R1, R5
  I_{j+1}: ADD R5, R1, R2
  - In any situation with a chance that I_{j+1} may finish before I_j, it must be ensured that the result of register R5 is not stored (written) before I_j has had a chance to fetch the operands (read).
  - May happen in out-of-order execution

Data Hazards: WAW

- A write after write (WAW) data hazard may occur in a concurrent execution environment

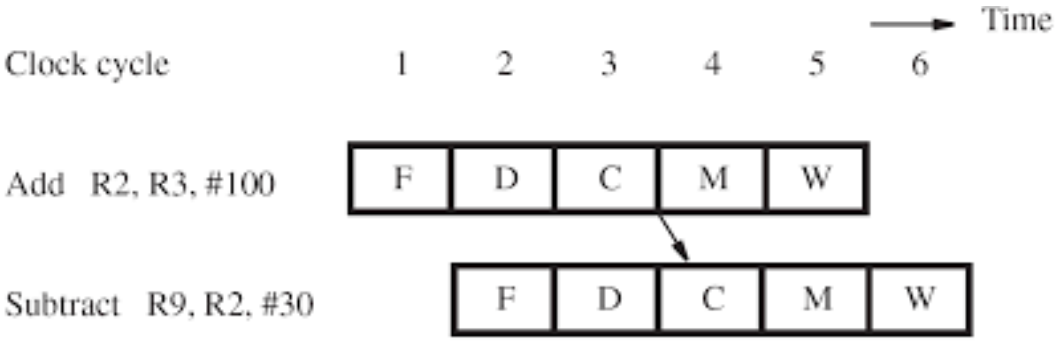
- Ex. I_j: ADD R2, R4, R7
  $I_{j+1:}$ ADD R2, R1, R3

  – The write back (write) of I_{j+1} must be delayed until I_j finishes executing.

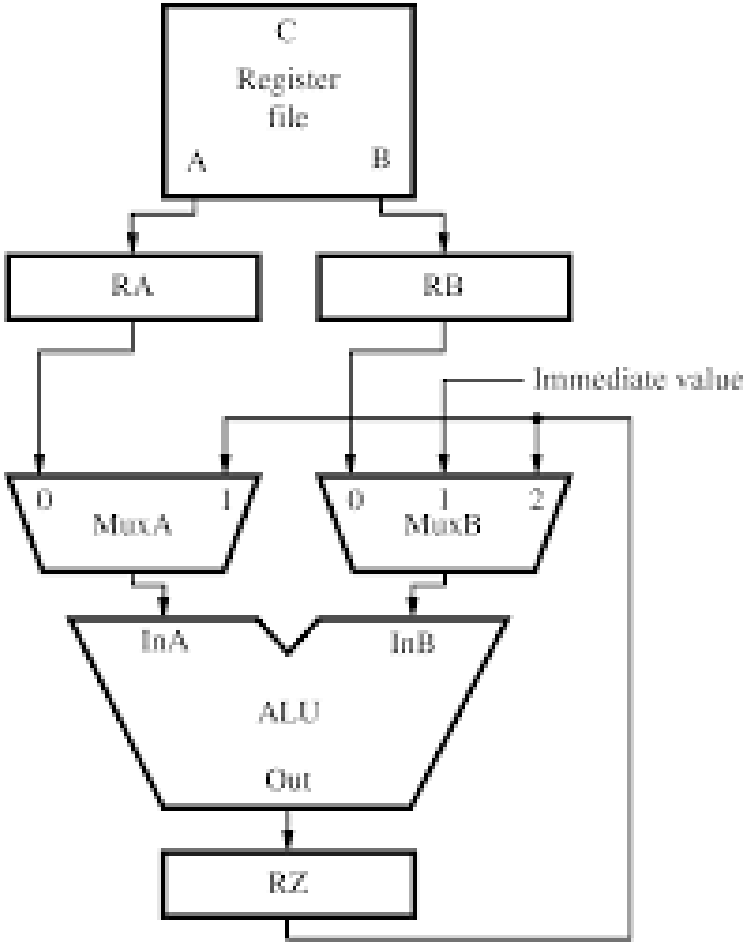
Operand forwarding

- Pipeline stalls due to data dependencies can be alleviated through the use of operand forwarding.
  - Consider the pair of instructions discussed above, where the pipeline is stalled for three cycles to enable the Subtract instruction to use the new value in register R2.
  - The desired value is actually available at the end of cycle 3, when the ALU completes the operation for the Add instruction.
  - This value is loaded into register RZ which is a part of interstage buffer B3.
  - Rather than stall the Subtract instruction, the hardware can forward the value from register RZ to where it is needed in cycle 4, which is the ALU input.

Operand forwarding



- A new multiplexer, MuxA, is inserted before input InA of the ALU, and the existing multiplexer MuxB is expanded with another input.
- The multiplexers select either a value read from the register file in the normal manner, or the value available in register RZ.


C
Register
file
A
B
RA
RB
Immediate value
0
MuxA
1
0
MuxB
1
2
InA
InB
ALU
Out
RZ