Chapter 8

Topics covered:
Pipelining

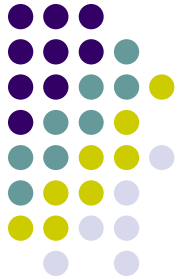
Basic concepts

- Speed of execution of programs can be improved in two ways:
  - Faster circuit technology to build the processor and the memory.
  - Arrange the hardware so that a number of operations can be performed simultaneously. The number of operations performed per second is increased although the elapsed time needed to perform any one operation is not changed.
- Pipelining is an effective way of organizing concurrent activity in a computer system to improve the speed of execution of programs.

Overview

- Pipelining is widely used in modern processors.
- Pipelining improves system performance in terms of **Throughput**. (The latency of each instruction still remains unchanged)
  - Throughput = No. of Instructions completed per unit time! Pipeline Improves Throughput, because multiple instructions can now run concurrently/ simultaneously!!! In **Ideal case** (with no Hazard): Throughput approaches 1 Instruction/cycle
- Pipelined organization requires sophisticated compilation techniques.

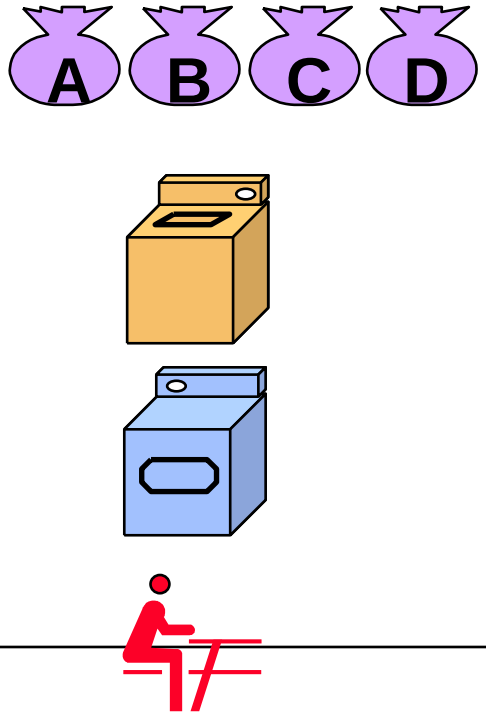
Making the Execution of Programs Faster



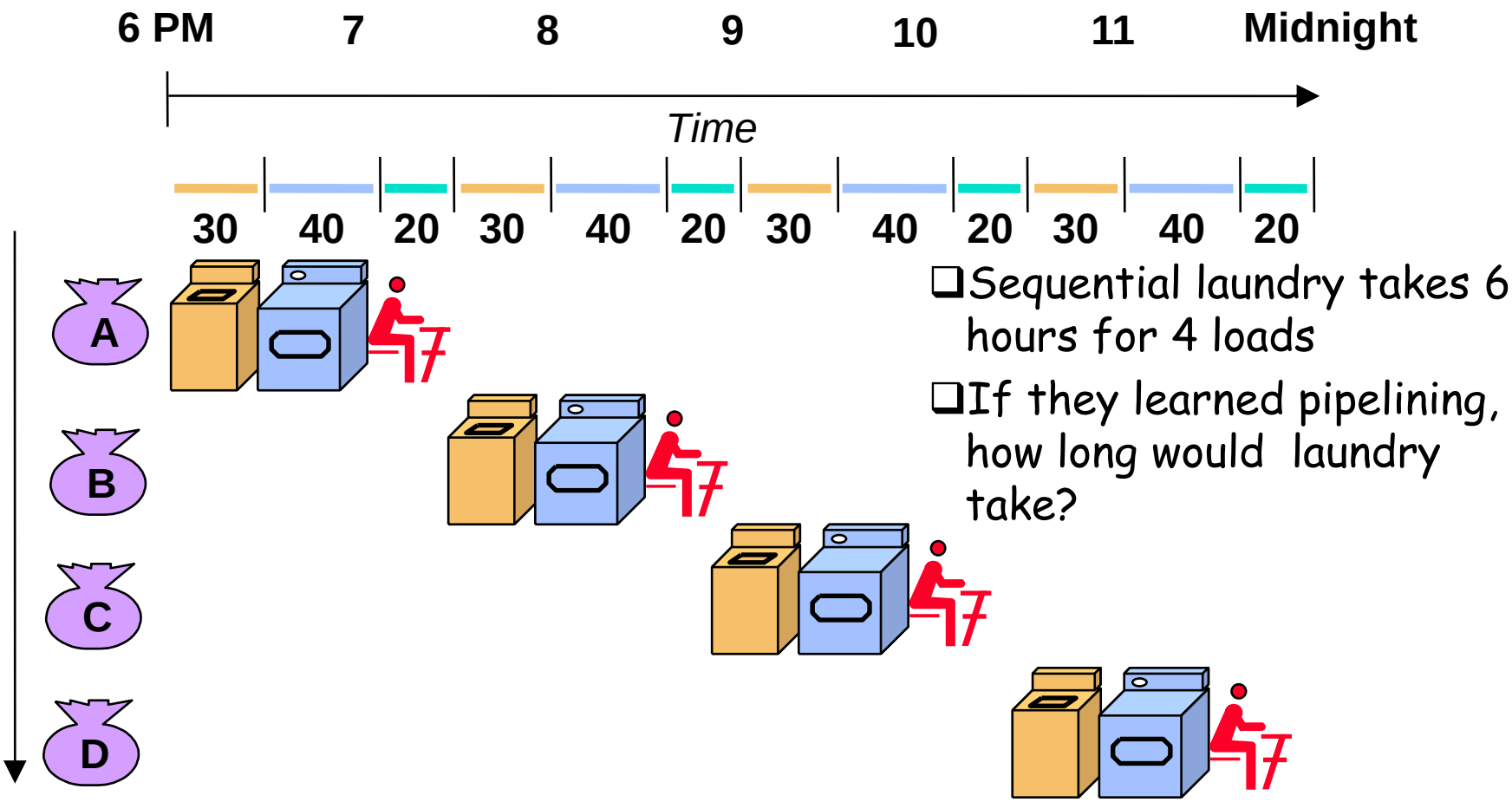
- Use **Faster circuit technology** to build the processor and the main memory. (Faster Clock Rate means Shorter Clock Cycle!)
- Arrange the hardware so that more than one operation can be performed in ***Parallel*** / at the same time. (This is the Pipelining!)
- In the latter way, the Throughput (number of operations performed per second) is increased, even though the Latency (Elapsed time needed to perform any one operation) is not changed.

Traditional Pipeline Concept

- Laundry Example
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- "Folder" takes 20 minutes

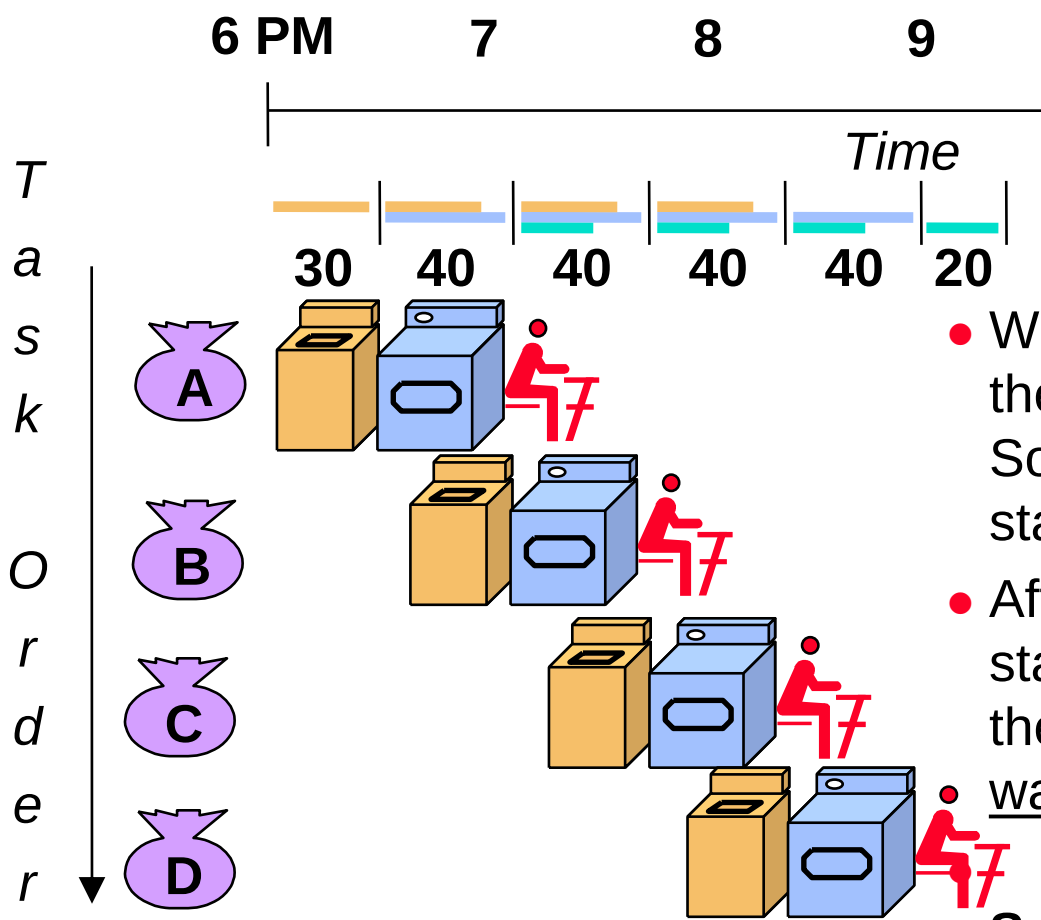


Traditional Pipeline Concept



- Sequential laundry takes 6 hours for 4 loads
- If they learned pipelining, how long would laundry take?

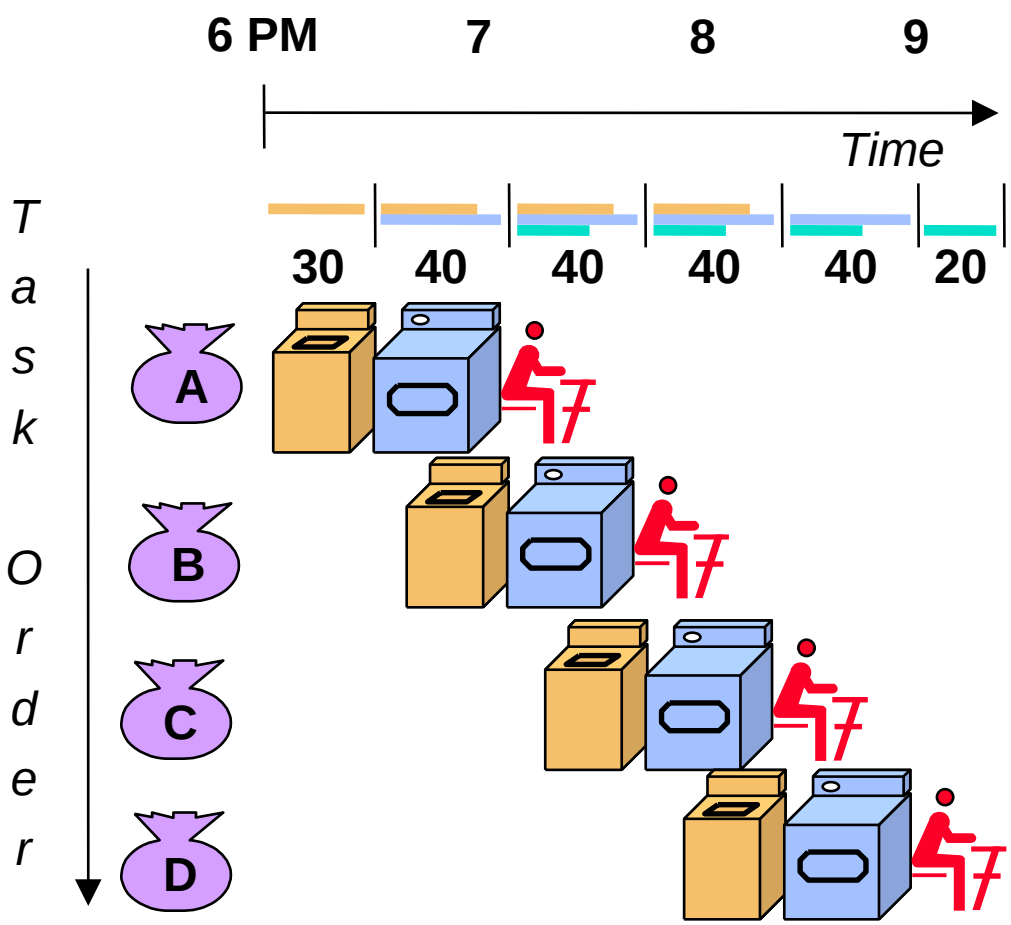
Traditional Pipeline Concept



Pipelined laundry takes 3.5 hours for 4 loads

- When Ann Puts her dress from the Washer to the Dryer, Someone else (say, Brian) can start using the Washer.
- After Ann leaves the Dryer and starts Folding, Brian can Move to the Dryer (from washer, after waiting for 10 minutes) and Cathy can start using the Washer!
- **So, Pipelining = Concurrent Operations!**

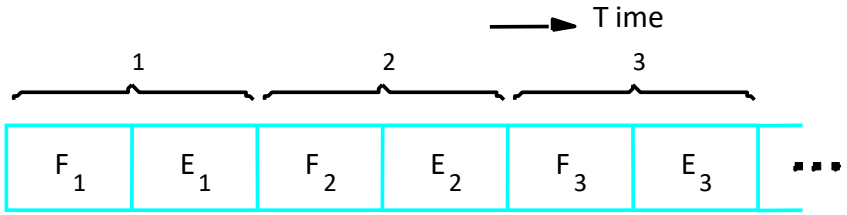
Traditional Pipeline Concept



- Pipelining doesn't help **latency** of single task, it helps **throughput** of entire workload
- Pipeline rate limited by slowest pipeline stage Dryer=**40 minutes**
- Multiple tasks simultaneously operating using different resource
- Potential speedup = Number of pipeline stages
- Unbalanced lengths of pipe stages reduces speedup (**40 min.)**
- Time to "fill" pipeline and time to "drain" it reduces speedup
- **Stall for Dependences**

Basic concepts (contd..)

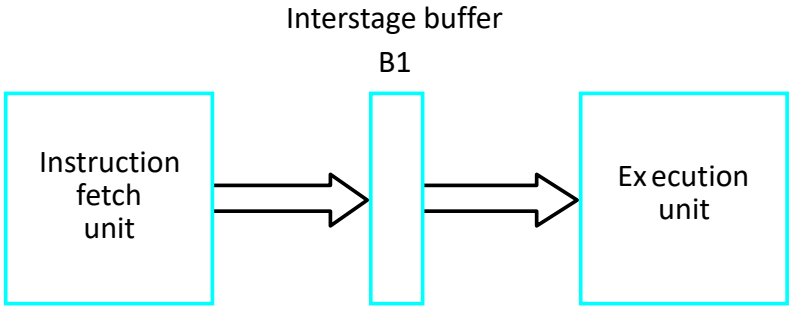
- *Processor executes a program by fetching and executing instructions one after the other.*
- *This is known as sequential execution.*
- *If F_i refers to the fetch step, and E_i refers to the execution step of instruction I_i, then sequential execution looks like:*



What if the execution of one instruction is overlapped with the fetching of the next one?

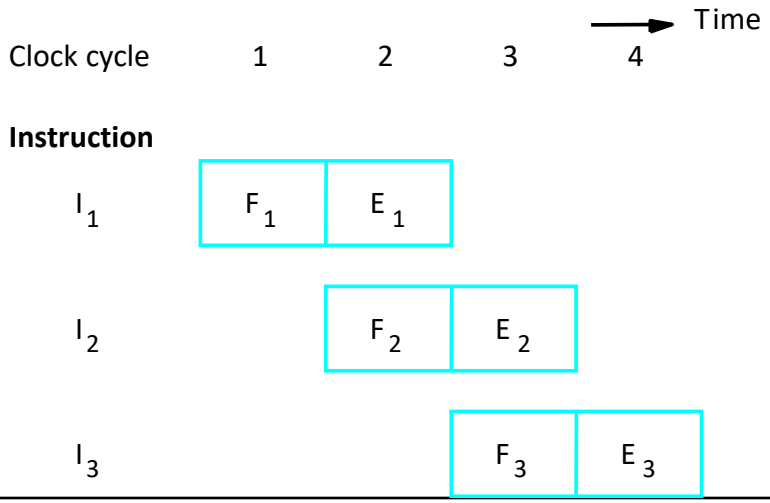
Basic concepts (contd..)

- *Computer has two separate hardware units, one for fetching instructions and one for executing instructions.*
- *Instruction is fetched by instruction fetch unit and deposited in an intermediate buffer B1.*
- *Buffer enables the instruction execution unit to execute the instruction while the fetch unit is fetching the next instruction.*
- *Results of the execution are deposited in the destination location specified by the instruction.*



Basic concepts (contd..)

- *Computer is controlled by a clock whose period is such that the fetch and execute steps of any instruction can be completed in one clock cycle.*
- *First clock cycle:*
  - *Fetch unit fetches an instruction I_1 (F_1) and stores it in B1.*
- *Second clock cycle:*
  - *Fetch unit fetches an instruction I_2 (F_2) , and execution unit executes instruction I_1 (E_1).*
- *Third clock cycle:*
  - *Fetch unit fetches an instruction I_3 (F_3), and execution unit executes instruction I_2 (E_2).*
- *Fourth clock cycle:*
  - *Execution unit executes instruction I_3 (E_3).*

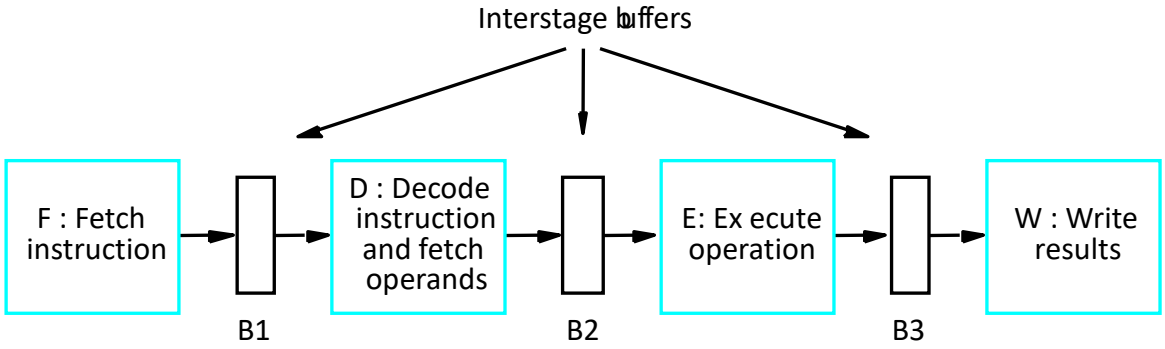


Basic concepts (contd..)

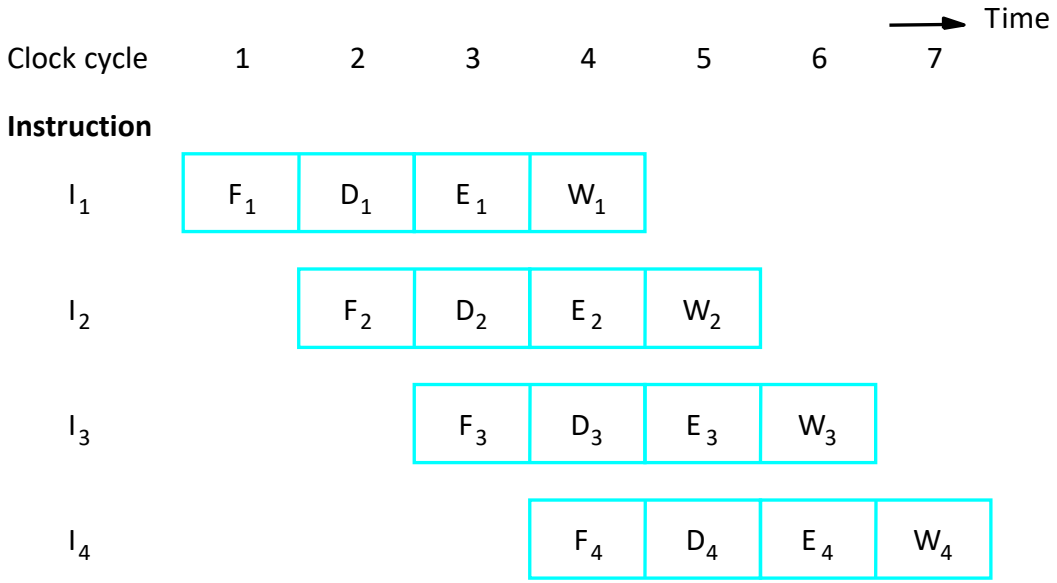
- In each clock cycle, the fetch unit fetches the next instruction, while the execution unit executes the current instruction stored in the interstage buffer.
  - Fetch and the execute units can be kept busy all the time.
- If this pattern of fetch and execute can be sustained for a long time, the completion rate of instruction execution will be twice that achievable by the sequential operation.
- Fetch and execute units constitute a two-stage pipeline.
  - Each stage performs one step in processing of an instruction.
  - Interstage storage buffer holds the information that needs to be passed from the fetch stage to execute stage.
  - New information gets loaded into the buffer every clock cycle.

Basic concepts (contd..)

- *Suppose the processing of an instruction is divided into four steps:*
  - *F Fetch: Read the instruction from the memory.*
  - *D Decode: Decode the instruction and fetch the source operands.*
  - *E Execute: Perform the operation specified by the instruction.*
  - *W Write: Store the result in the destination location.*
- *There are four distinct hardware units, for each one of the steps.*
- *Information is passed from one unit to the next through an interstage buffer.*
- *Three interstage buffers connecting four units.*
- *As an instruction progresses through the pipeline, the information needed by the downstream units must be passed along.*

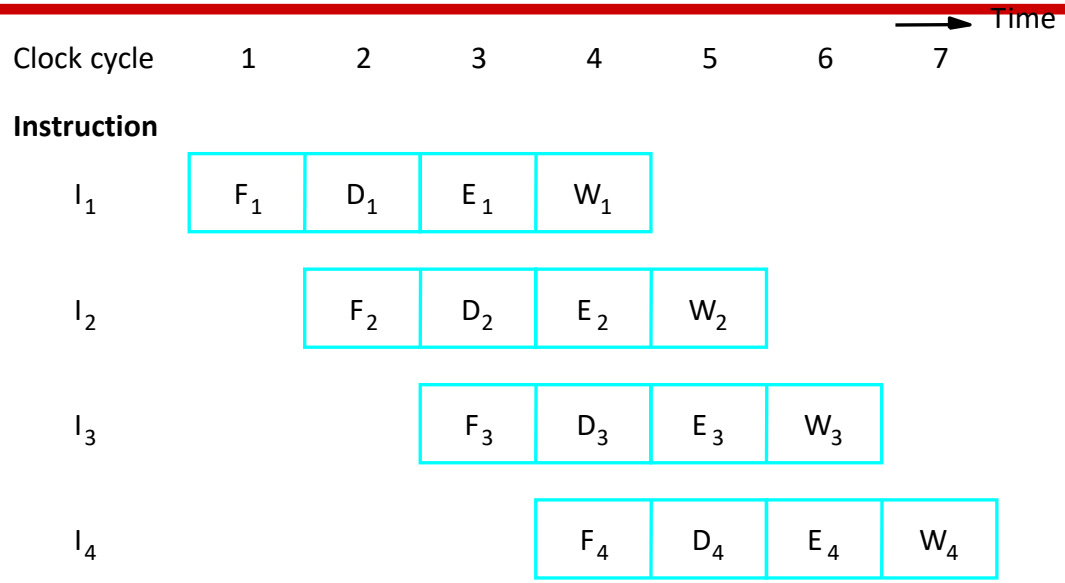


Basic concepts (contd..)



Clock cycle 1: F1
Clock cycle 2: D1, F2
Clock cycle 3: E1, D2, F3
Clock cycle 4: W1, E2, D3, F4
Clock cycle 5: W2, E3, D4
Clock cycle 6: W3, E3, D4
Clock cycle 7: W4

Basic concepts (contd..)



During clock cycle #4:

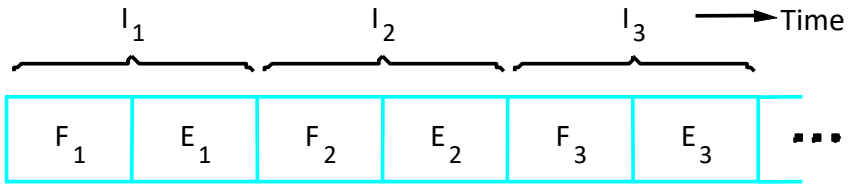
- *Buffer B1 holds instruction I_3, which is being decoded by the instruction-decoding unit. Instruction I_3 was fetched in cycle 3.*
- *Buffer B2 holds the source and destination operands for instruction I_2. It also holds the information needed for the Write step (W_2) of instruction I_2. This information will be passed to the stage W in the following clock cycle.*
- *Buffer B1 holds the results produced by the execution unit and the destination information for instruction I_1.*

Pipelining: How Implemented in a Computer? Also: How Pipelining Improves Throughput, but not Latency

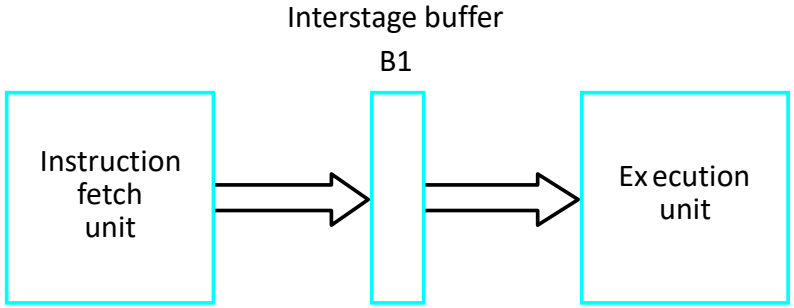
Throughput: No. of instruction completed per unit time

Latency: no. of clock cycles elapsed between the starting and ending of an instruction

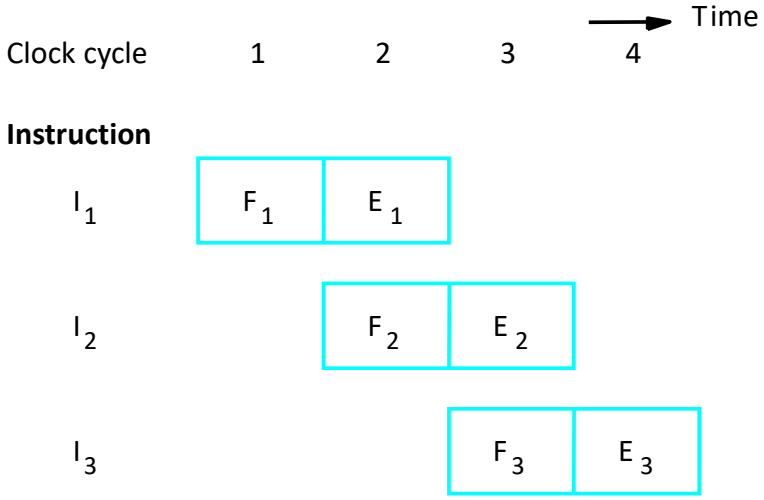
Suppose: Instruction = Fetch + Execution



(a) Sequential execution



(b) Hardware organization

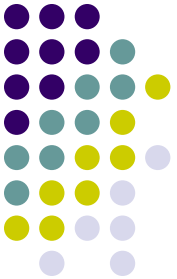


(c) Pipelined execution

Fig. 8.1. Basic idea of instruction pipelining

Note: Sequential Execution (fig. a) => Throughput = 0.5 instruction/cycle;
Pipelined Execution (fig. c) => (approx.) 1 instruction/cycle (starting from 2nd cycle) In both Cases: Latency of each instruction is 2 cycles

Use the Idea of Pipelining in a Computer



Instruction = Fetch + Decode + Execution + Write

"**Fetch**" stages fetches the next instruction from the cache or main memory

"**Decode**" stage prepares and fetches the source the operands

"**Execution**" => The Actual Execution, May involve ALU for Add, Sub, Mul, Div, And, OR, Shift etc ...

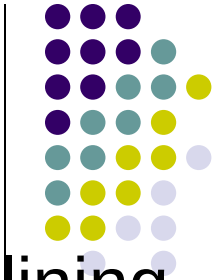
"**Write**" stage saves/stores the result, such as writing to a Register or Memory location

Here: Throughput => (approx.) 1 instruction/cycle (starting from the 4th cycle). And, Latency: 4 cycles for each instruction

Role of cache memory

- Each stage in the pipeline is expected to complete its operation in one clock cycle:
  - Clock period should be sufficient to complete the longest task.
  - Units which complete the tasks early remain idle for the remaining clock period.
  - Tasks being performed in different stages should require about the same amount of time for pipelining to be effective.
- If instructions are to be fetched from the main memory, the instruction fetch stage would take as much as ten times greater than the other stage operations inside the processor.
- However, if instructions are to be fetched from the cache memory which is on the processor chip, the time required to fetch the instruction would be more or less similar to the time required for other basic operations.

Pipeline Performance

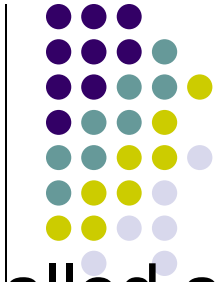


- The potential increase in performance from pipelining is ***proportional*** to the number of pipeline stages, say ***n***.
- ***n*** stages => ***n*** instructions can run simultaneously, and the clock pulse can be (ideally) as small as just **1/n-th of the previous pulse length**!!!
- Ideally, the Throughput is Increased ***n*-times!!!**
- However, this increase would be achieved only if all pipeline stages require the same time to complete, and there is no interruption (hazard) throughout program execution.
- Unfortunately, this is not true.

Pipeline performance

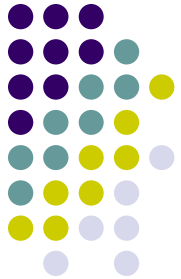
- Potential increase in performance achieved by using pipelining is proportional to the number of pipeline stages.
  - For example, if the number of pipeline stages is 4, then the rate of instruction processing is 4 times that of sequential execution of instructions.
  - Pipelining does not cause a single instruction to be executed faster, it is the throughput that increases.
- This rate can be achieved only if the pipelined operation can be sustained without interruption through program instruction.
- If a pipelined operation cannot be sustained without interruption, the pipeline is said to "stall".
- A condition that causes the pipeline to stall is called a "hazard".

Pipeline Performance



- Any condition that causes a pipeline to stall is called a **Hazard**
- Data hazard – any condition in which either the source or the destination operands of an instruction are not available at the time expected in the pipeline. So some operation has to be delayed, and the pipeline stalls.
- Instruction (control) hazard – a delay in the availability of an instruction causes the pipeline to stall.
- Structural hazard – the situation when two instructions require the use of a given hardware resource at the same time.

Pipeline Performance



Instruction hazard

(example of How Cache Miss causes Instruction Hazard and the Pipelins Stalls)

Idle periods – stalls (bubbles)

Pipeline Performance

I_2: Load X(R1), R2 **I_3: MOV R3, R2**

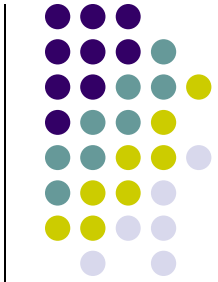
Structural hazard

The Execution Stage E_2 Computes X+R1

Then An Extra Memory Access Stage, M_2 is Required to Fetch the source Operand from the Memory Address [X+R1]

Structural Hazard, because the "Write" unit Can't Perform Two different Writes in One clock Pulse

Pipeline Performance



- Again, pipelining does not result in individual instructions being executed faster; rather, it is the throughput that increases.
- Throughput is measured by the rate at which instruction execution is completed.
- Pipeline stall causes degradation in pipeline performance.
- We need to identify all hazards that may cause the pipeline to stall and to find ways to minimize their impact.

Data Hazards

- We must ensure that the results obtained when instructions are executed in a pipelined processor are identical to those obtained when the same instructions are executed sequentially.
- Hazard occurs

  $A \leftarrow 3 + R_1$

  $B \leftarrow 4 \times A$
- No hazard

  $A \leftarrow 5 \times C$

  $B \leftarrow 20 + C$
- When two operations depend on each other, they must be executed sequentially in the correct order.
- Another example:

  Mul R2, R3, R4 ➔ R4 = R2 + R3

  Add R5, R4, R6 ➔ R6 = R4 + R5

Data Hazards

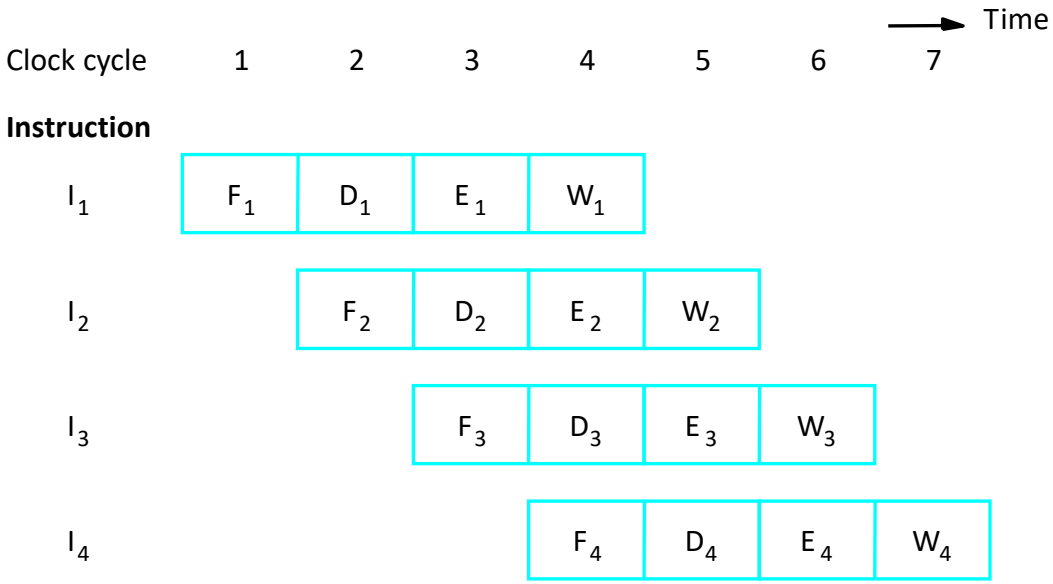
Mul R2, R3, R4

Add R5, R4, R6

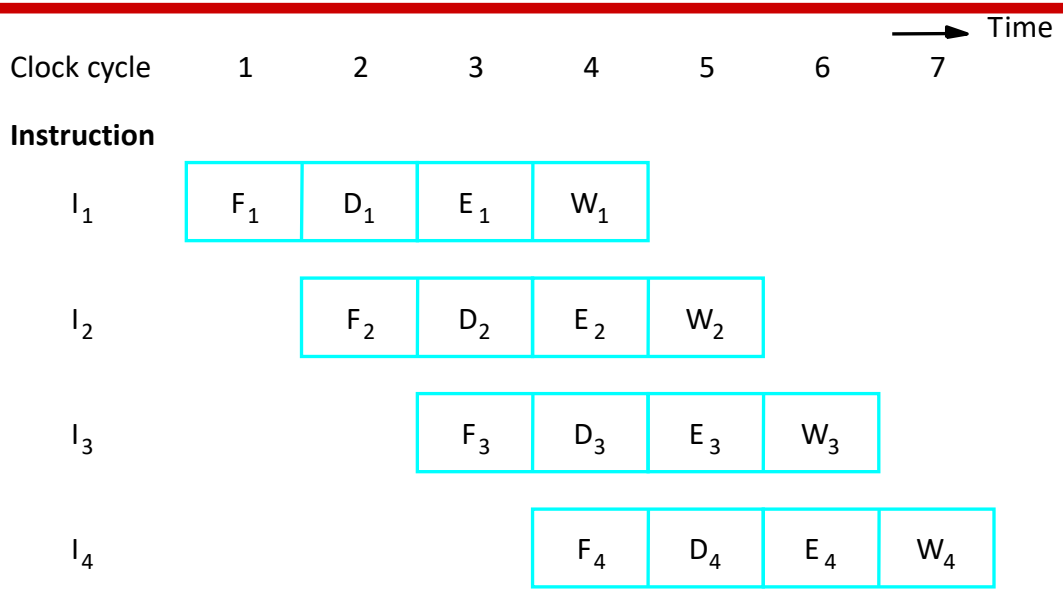
Figure 8.6. Pipeline stalled by data dependency between D_2 and W_1

Data hazard

- *Execution of the instruction occurs in the E stage of the pipeline.*
- *Execution of most arithmetic and logic operations would take only one clock cycle.*
- *However, some operations such as division would take more time to complete.*
- *For example, the operation specified in instruction I2 takes three cycles to complete from cycle 4 to cycle 6.*



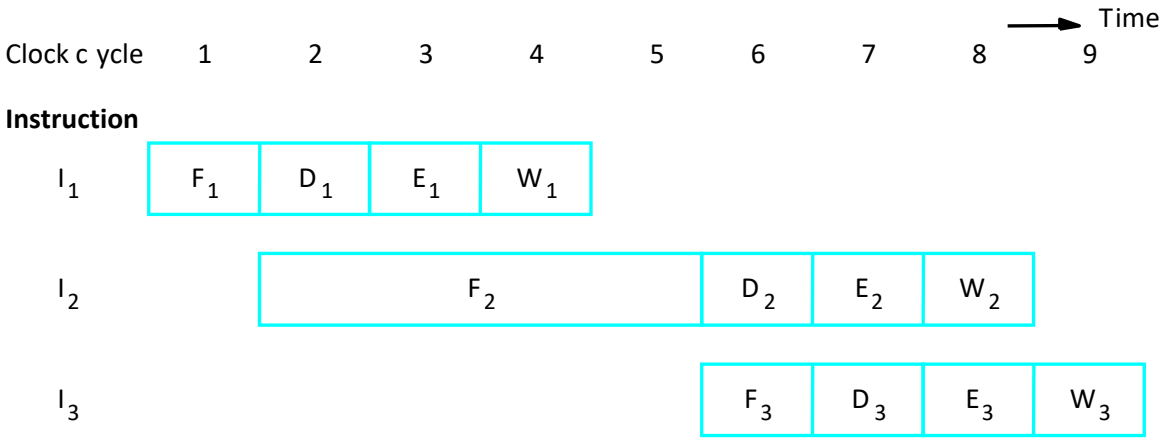
Data hazard (contd..)



- *Cycles 5 and 6, the Write stage is idle, because it has no data to work with.*
- *Information in buffer B2 must be retained till the execution of the instruction I_2 is complete.*
- *Stage 2, and by extension stage 1 cannot accept new instructions because the information in B1 cannot be overwritten.*
- *Steps D_6 and F_5 must be postponed.*
- *A data hazard is a condition in which either the source or the destination operand is not available at the time expected in the pipeline.*

Control or instruction hazard

- *Pipeline may be stalled because an instruction is not available at the expected time.*
- *For example, while fetching an instruction a cache miss may occur, and hence the instruction may have to be fetched from the main memory.*
- *Fetching the instruction from the main memory takes much longer than fetching the instruction from the cache.*
- *Thus, the fetch cycle of the instruction cannot be completed in one cycle.*
- *For example, the fetching of instruction I_2 results in a cache miss.*
- *Thus, F_2 takes 4 clock cycles instead of 1.*



Control or instruction hazard (contd..)

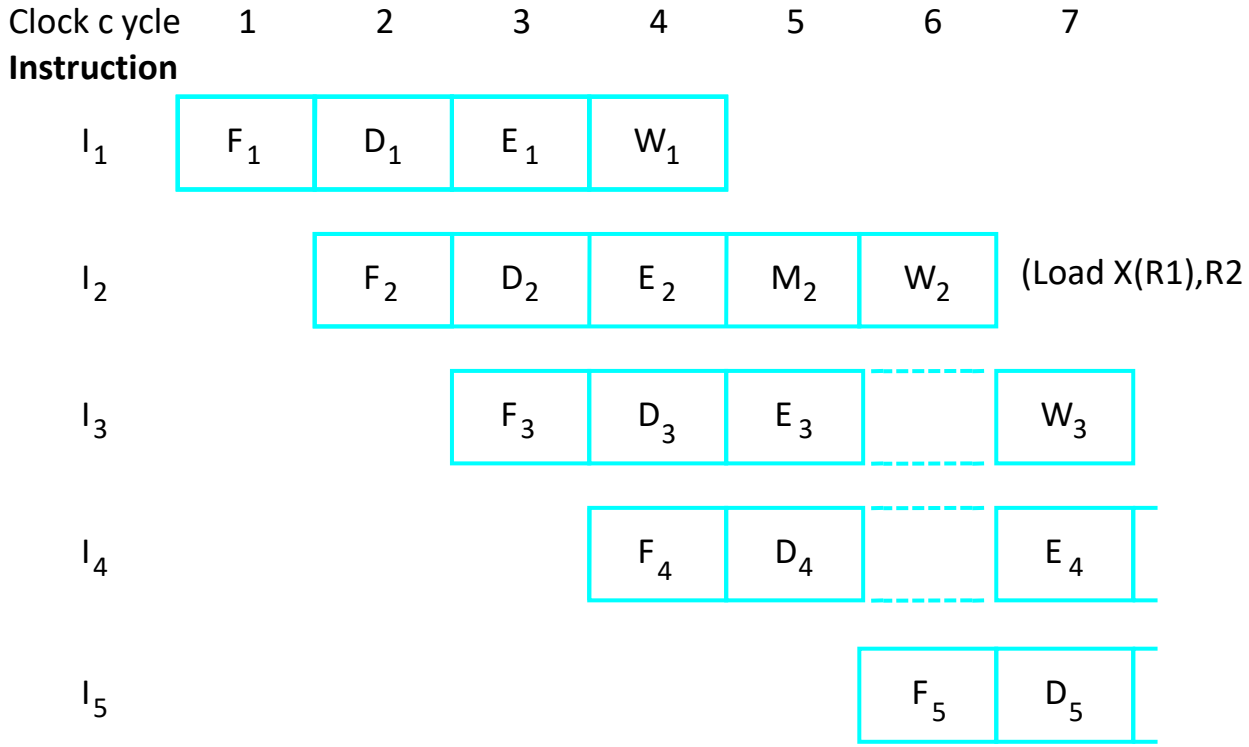
- *Fetch operation for instruction I2 results in a cache miss, and the instruction fetch unit must fetch this instruction from the main memory.*
- *Suppose fetching instruction I2 from the main memory takes 4 clock cycles.*
- *Instruction I2 will be available in buffer B1 at the end of clock cycle 5.*
- *The pipeline resumes its normal operation at this point.*
- *Decode unit is idle in cycles 3 through 5.*
- *Execute unit is idle in cycles 4 through 6.*
- *Write unit is idle in cycles 5 through 7.*
- *Such idle periods are called as stalls or bubbles.*
- *Once created in one of the pipeline stages, a bubble moves downstream unit it reaches the last unit.*

Clock c ycle → Time	1	2	3	4	5	6	7	8	9
Stage									
F: Fetch	F_1	F_2	F_2	F_2	F_2	F_3			
D: Decode		D_1	idle	idle	idle	D_2	D_3		
E: Execute			E_1	idle	idle	idle	E_2	E_3	
W: Write				W_1	idle	idle	idle	W_2	W_3

Structural hazard

- Two instructions require the use of a hardware resource at the same time.
- Most common case is in access to the memory:
  - One instruction needs to access the memory as part of the Execute or Write stage.
  - Other instruction is being fetched.
  - If instructions and data reside in the same cache unit, only one instruction can proceed and the other is delayed.
- Many processors have separate data and instruction caches to avoid this delay.
- In general, structural hazards can be avoided by providing sufficient resources on the processor chip.

Structural hazard (contd..)



- *Memory address X+R1 is computed in step E_2 in cycle 4, memory access takes place in cycle 5, operand read from the memory is written into register R2 in cycle 6.*
- *Execution of instruction I_2 takes two clock cycles 4 and 5.*
- *In cycle 6, both instructions I_2 and I_3 require access to register file.*
- *Pipeline is stalled because the register file cannot handle two operations at once.*

Pipelining and performance

- Pipelining does not cause an individual instruction to be executed faster, rather, it increases the throughput.
  - Throughput is defined as the rate at which instruction execution is completed.
- When a hazard occurs, one of the stages in the pipeline cannot complete its operation in one clock cycle.
  - The pipeline stalls causing a degradation in performance.
- Performance level of one instruction completion in each clock cycle is the upper limit for the throughput that can be achieved in a pipelined processor.

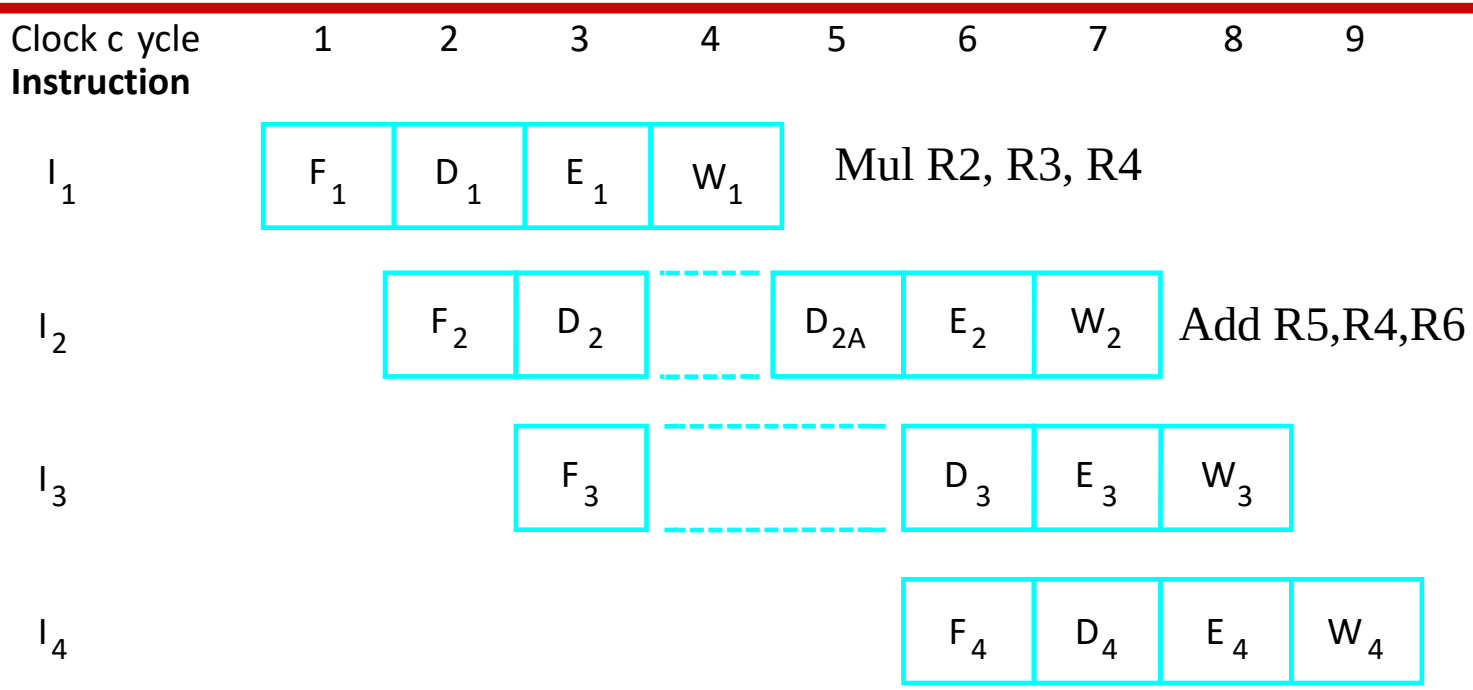
Data hazards

- *Data hazard is a situation in which the pipeline is stalled because the data to be operated on are delayed.*
- *Consider two instructions:*

$$I_1: A = 3 + A$$
$$I_2: B = 4 \times A$$

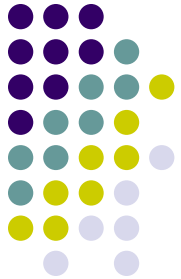
- *If $A = 5$, and I_1 and I_2 are executed sequentially, $B=32$.*
- *In a pipelined processor, the execution of I_2 can begin before the execution of I_1.*
- *The value of A used in the execution of I_2 will be the original value of 5 leading to an incorrect result.*
- *Thus, instructions I_1 and I_2 depend on each other, because the data used by I_2 depends on the results generated by I_1.*
- *Results obtained using sequential execution of instructions should be the same as the results obtained from pipelined execution.*
- *When two instructions depend on each other, they must be performed in the correct order.*

Data hazards (contd..)



- *Mul instruction places the results of the multiply operation in register R4 at the end of clock cycle 4.*
- *Register R4 is used as a source operand in the Add instruction. Hence the Decode Unit decoding the Add instruction cannot proceed until the Write step of the first instruction is complete.*
- *Data dependency arises because the destination of one instruction is used as a source in the next instruction.*

Handling Data Hazard in Hardwire: Operand Forwarding

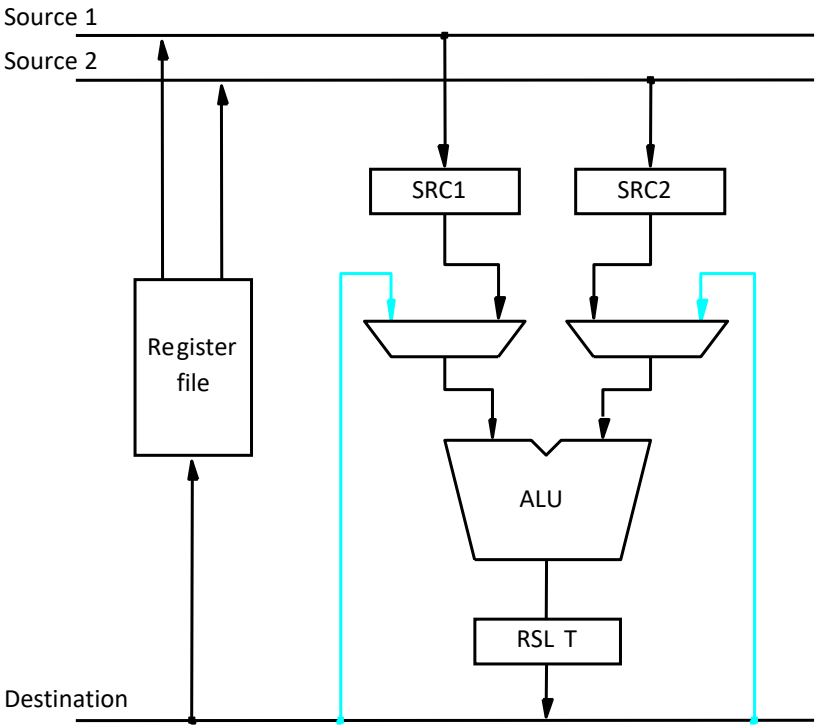


- Instead of from the register file, the second instruction can get data directly from the output of ALU immediately just after the “E” (i.e., execution) stage of the previous instruction is completed.
- A special arrangement needs to be made to “forward” the output of ALU to the input of ALU.

Operand forwarding

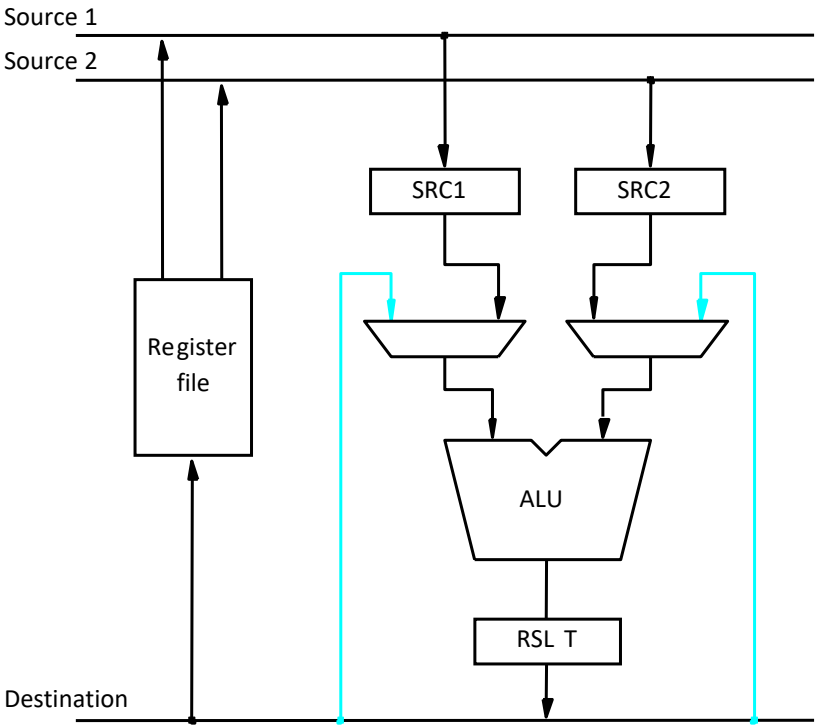
- *Data hazard occurs because the destination of one instruction is used as the source in the next instruction.*
- *Hence, instruction I_2 has to wait for the data to be written in the register file by the Write stage at the end of step W_1.*
- *However, these data are available at the output of the ALU once the Execute stage completes step E_1.*
- *Delay can be reduced or even eliminated if the result of instruction I1 can be forwarded directly for use in step E_2.*
- *This is called "operand forwarding".*

Operand forwarding (contd..)



- *Similar to the three-bus organization.*
- *Registers SRC1, SRC2 and RSLT have been added.*
- *SRC1, SRC2 and RSLT are interstage buffers for pipelined operation.*
- *SRC1 and SRC2 are part of buffer B2.*
- *RSLT is part of buffer B3.*
- *Data forwarding mechanism is shown by the two red lines.*
- *Two multiplexers connected at the inputs to the ALU allow the data on the destination bus to be selected instead of the contents of SRC1 and SRC2 register.*

Operand forwarding (contd..)



I_1: *Mul R2, R3, R4*
I_2: *Add R5, R4, R6*

Clock cycle 3:
- *Instruction I_2 is decoded, and a data dependency is detected.*
- *Operand not involved in the dependency, register R5 is loaded in register SRC1.*

Clock cycle 4:
- *Product produced by I_1 is available in register RSLT.*
- *The forwarding connection allows the result to be used in step E_2.*

Instruction I_2 proceeds without interruption.

Handling data dependency in software

- Data dependency may be detected by the hardware while decoding the instruction:
  - Control hardware may delay by an appropriate number of clock cycles reading of a register till its contents become available. The pipeline stalls for that many number of clock cycles.
- Detecting data dependencies and handling them can also be accomplished in software.
  - Compiler can introduce the necessary delay by introducing an appropriate number of NOP instructions. For example, if a two-cycle delay is needed between two instructions then two NOP instructions can be introduced between the two instructions.

I_1: *Mul R2, R3, R4*
NOP
NOP
I_2: *Add R5, R4, R6*

Superscalar operation [SELF STUDY]

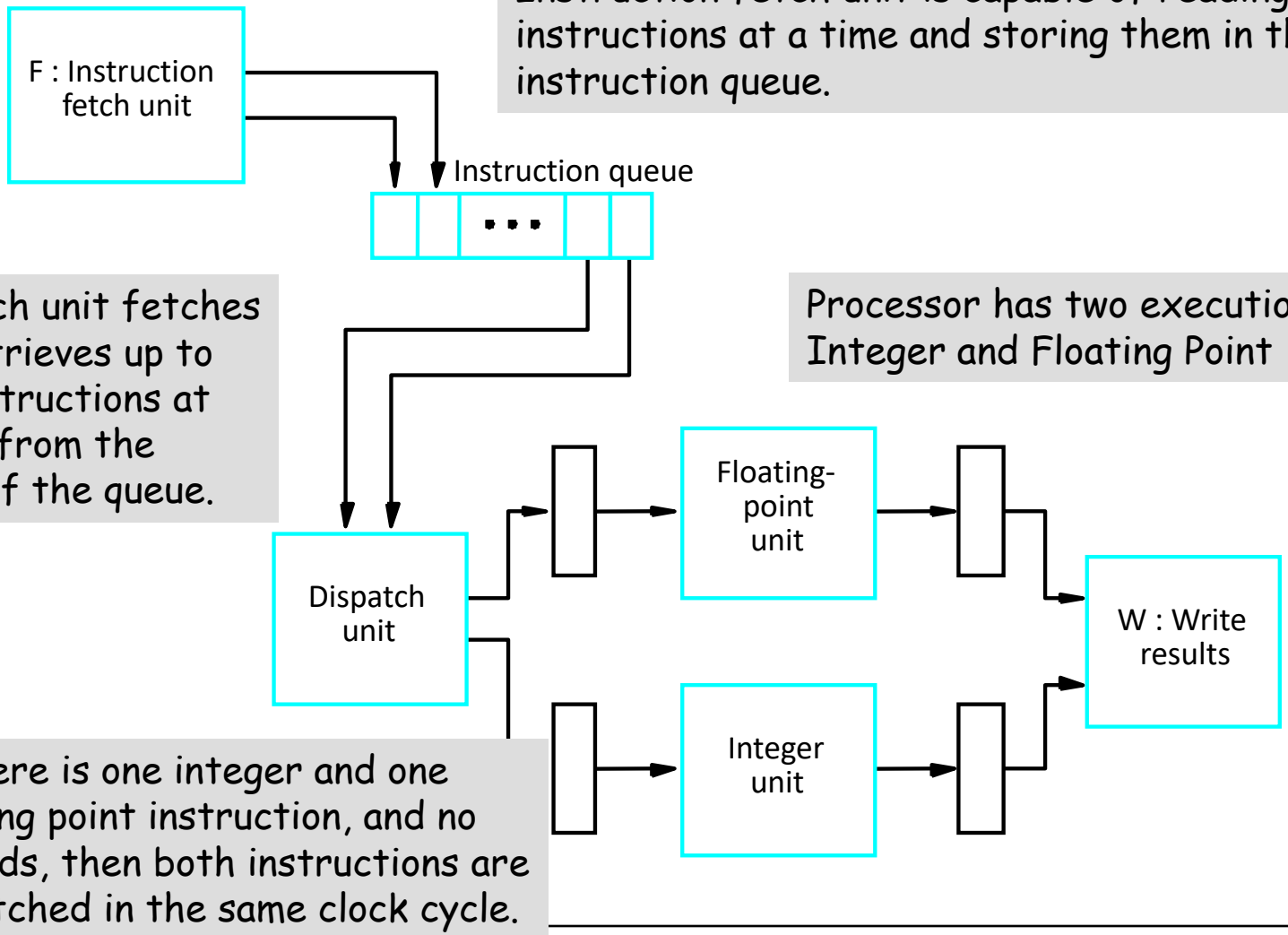
- Pipelining enables multiple instructions to be executed concurrently by dividing the execution of an instruction into several stages:
  - Instructions enter the pipeline in strict program order.
  - If the pipeline does not stall, one instruction enters the pipeline and one instruction completes execution in one clock cycle.
  - Maximum throughput of a pipelined processor is one instruction per clock cycle.
- An alternative approach is to equip the processor with multiple processing units to handle several instructions in parallel in each stage.

Superscalar operation (contd..)

- If a processor has multiple processing units then several instructions can start execution in the same clock cycle.
  - Processor is said to use "multiple issue".
- These processors are capable of achieving instruction execution throughput of more than one instruction per cycle.
- These processors are known as "superscalar processors".

Superscalar operation (contd..)

Instruction fetch unit is capable of reading two instructions at a time and storing them in the instruction queue.



Dispatch unit fetches and retrieves up to two instructions at a time from the front of the queue.

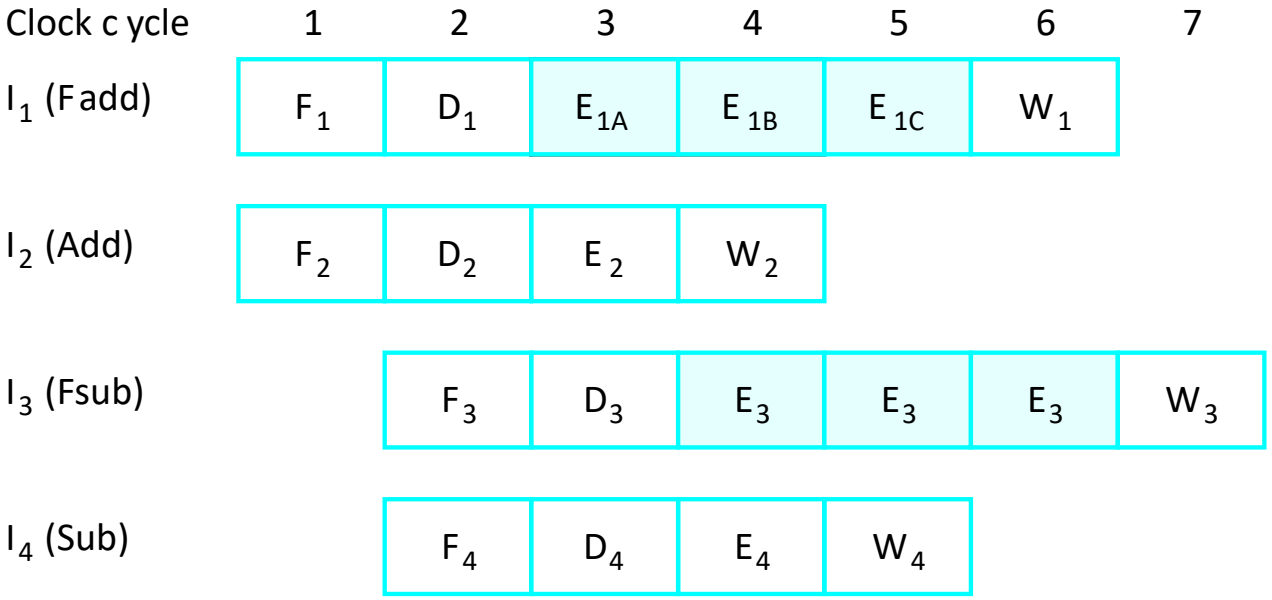
Processor has two execution units: Integer and Floating Point

If there is one integer and one floating point instruction, and no hazards, then both instructions are dispatched in the same clock cycle.

Superscalar operation (contd..)

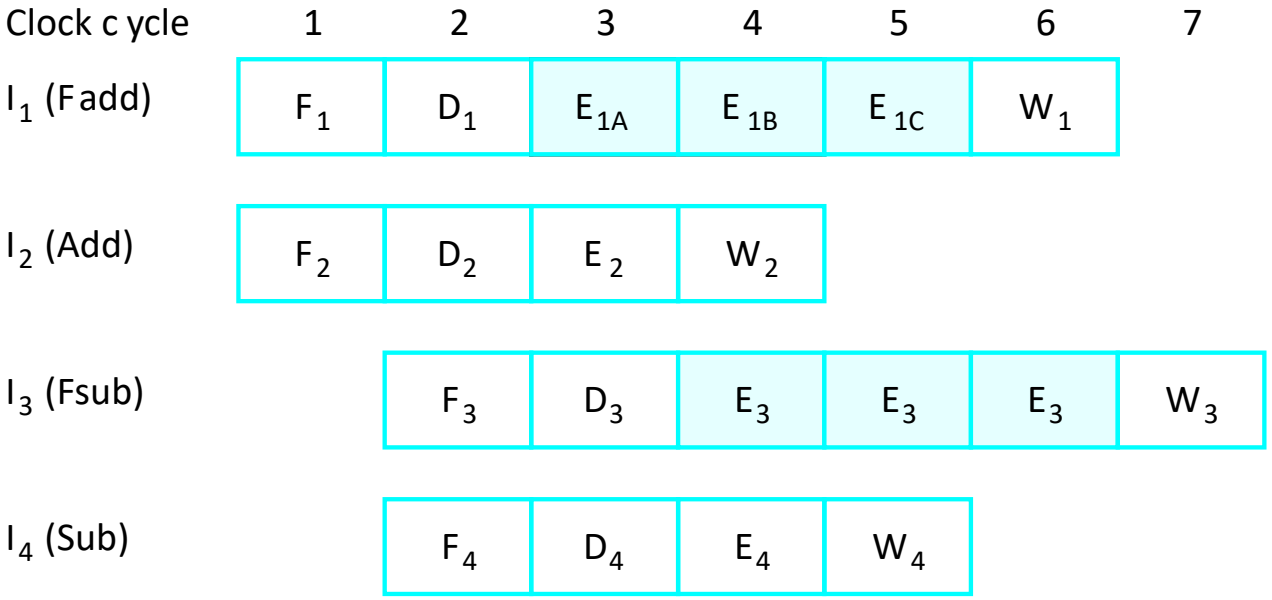
- Various hazards cause a even greater deterioration in performance in case of a superscalar processor.
- Compiler can avoid many hazards by careful ordering of instructions:
  - For example, the compiler should try to interleave floating-point and integer instructions.
  - Dispatch unit can then dispatch two instructions in most clock cycles, and keep both integer and floating point units busy most of the time.
- If the compiler can order instructions in such a way that the available hardware units can be kept busy most of the time, high performance can be achieved.

Superscalar operation (contd..)



- *Instructions in the floating-point unit take three cycles to execute.*
- *Floating-point unit is organized as a three-stage pipeline.*
- *Instructions in the integer unit take one cycle to execute.*
- *Integer unit is organized as a single-stage pipeline.*
- *Clock cycle 1:*
  - *Instructions I_1 (floating point) and I_2 (integer) are fetched.*
- *Clock cycle 2:*
  - *Instructions I_1 and I_2 are decoded and dispatched, I_3 is fetched.*

Superscalar operation (contd..)



•*Clock cycle 3:*

- I_1 and I_2 begin execution, I_2 completes execution. I_3 is dispatched to floating

- point unit and I_4 is dispatched to integer unit.

Clock cycle 4:

- I_1 continues execution, I_3 begins execution, I_2 completes Write stage, I_4 completes execution.

Clock cycle 5:

- I_1 completes execution, I_3 continues execution, and I_4 completes Write.

Order of completion is I_2, I_4, I_1, I_3