BCN 1043

COMPUTER ARCHITECTURE & ORGANIZATION

By
Dr. Mritha Ramalingam

Faculty of Computer Systems & Software Engineering
mritha@ump.edu.my

AUTHORS

- **Dr. Mohd Nizam Mohmad Kahar** (mnizam@ump.edu.my)
- **Jamaludin Sallim** (jamal@ump.edu.my)
- **Dr. Syafiq Fauzi Kamarulzaman** (syafiq29@ump.edu.my)
- **Dr. Mritha Ramalingam** (mritha@ump.edu.my)

Faculty of Computer Systems & Software Engineering

Learning outcomes

- Able to describe the basic building blocks of computers and their role in the historical development of computer architecture.
- Articulate the differences between single thread vs. multiple thread
- Articulate the concept of strong vs. weak scaling, i.e., how performance is affected by scale of problem vs. scale of resources to solve the problem.
- Design a simple logic circuit using the fundamental building blocks of logic design.
- Use tools for capture, synthesis, and simulation to evaluate a logic design. [Usage]
- Write a simple sequential problem and a simple parallel version of the same program.
- Evaluate performance of simple sequential and parallel versions of a program with different problem sizes, and be able to describe the speed-ups achieved.

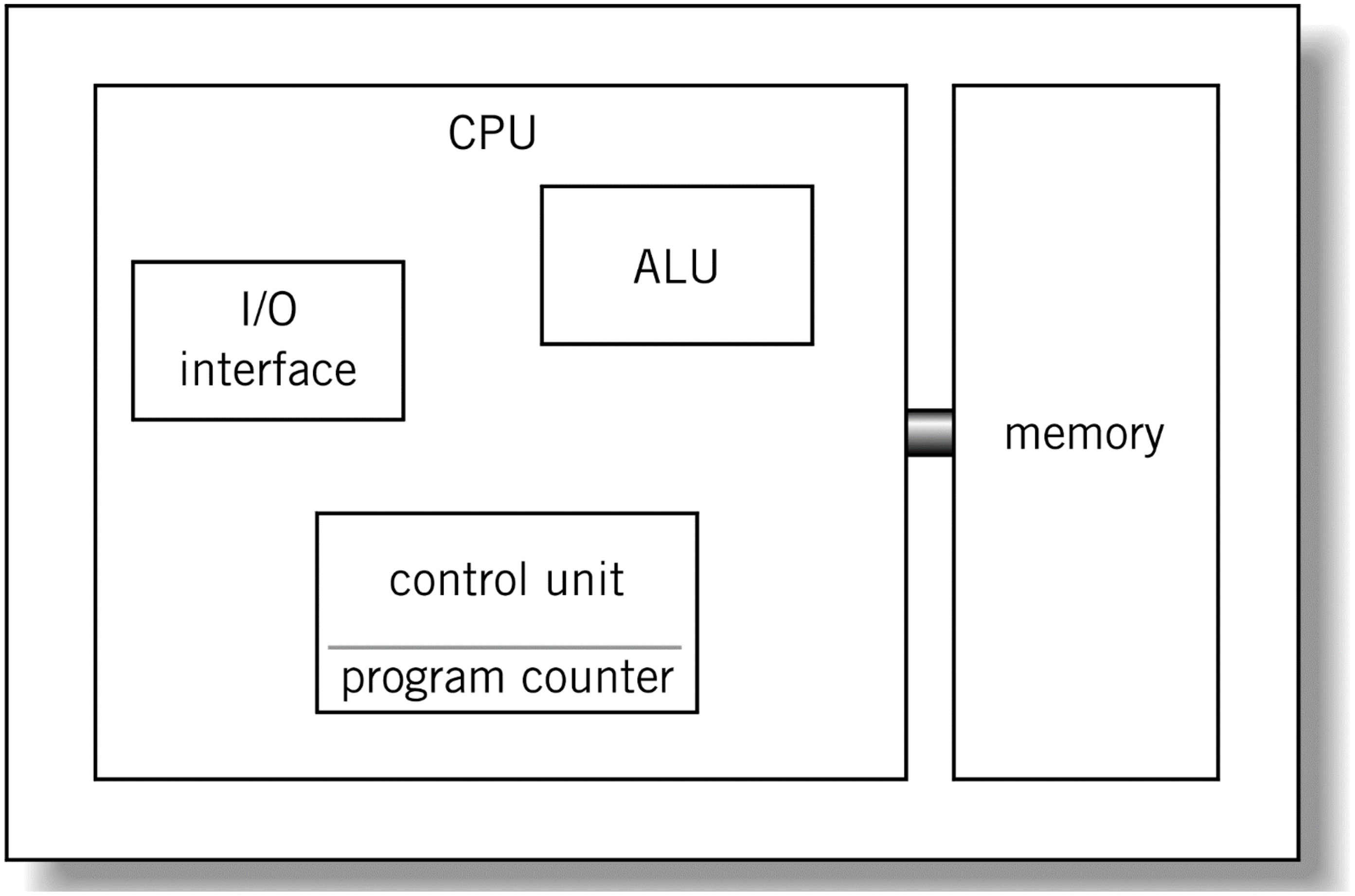
Computational Paradigms

I. Organization of Processor
II. organization of Register
III. Instruction cycle & Data Flow
IV. Instruction pipelining

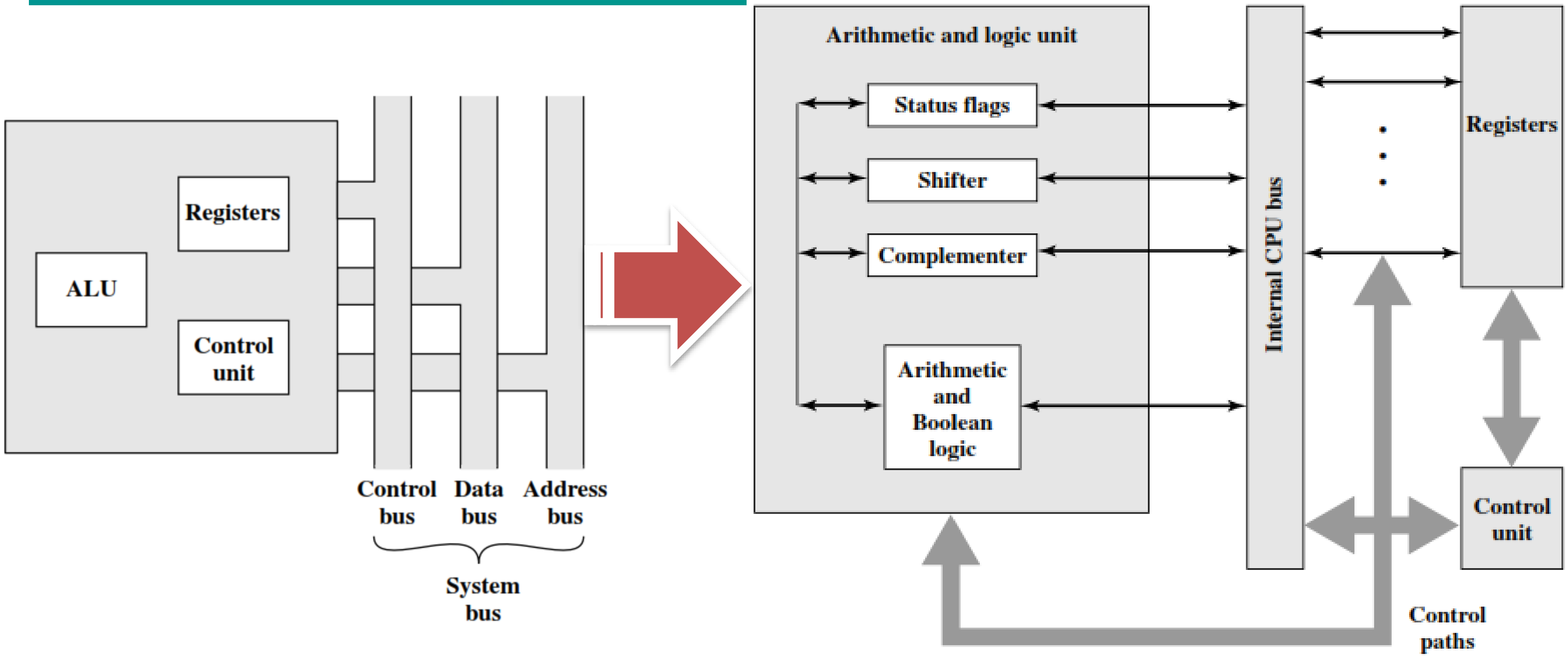
Organization of Processor

- arithmetic logic unit
- control unit
  - Subparts:
    - *Memory management unit*
    - *I/O Interface*
    - *Bust Interface Unit*
- *Registers*
  - Temporary storage elements

BLOCK DIAGRAM



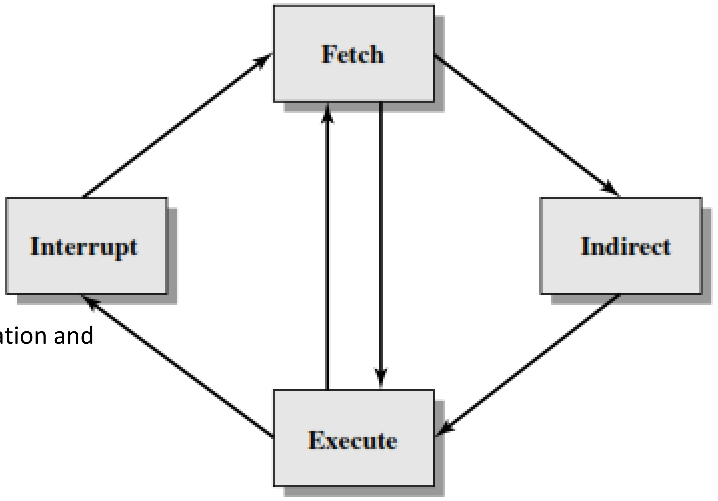
Internal view



Source: William Stallings, Computer Organization and Architecture,9th Edn

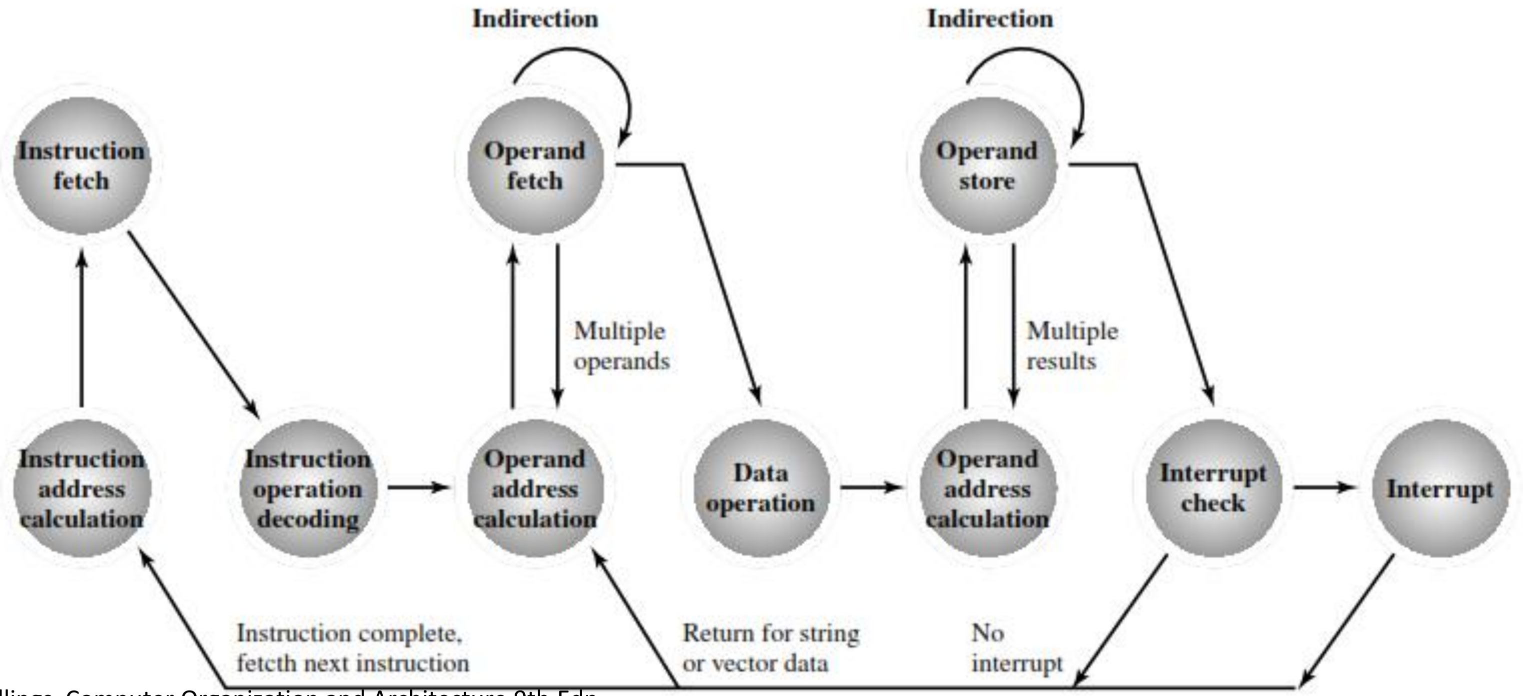
Instruction Cycle

- Fetch the instruction
  - Fetch the next instruction to be executed from memory
- Decode the instruction /Execute
  - Determine the opcode / perform the operation
- Interrupt
  - If an interrupt occurs, store the current state of process and attend the interrupt
- Indirect
  - expects additional memory access



Source: William Stallings, Computer Organization and Architecture,9th Edn

Instruction Cycle


Indirection
Indirection
Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data operation
Operand address calculation
Interrupt check
Interrupt
Instruction complete, fetcth next instruction
Return for string or vector data
No interrupt

Source: William Stallings, Computer Organization and Architecture,9th Edn

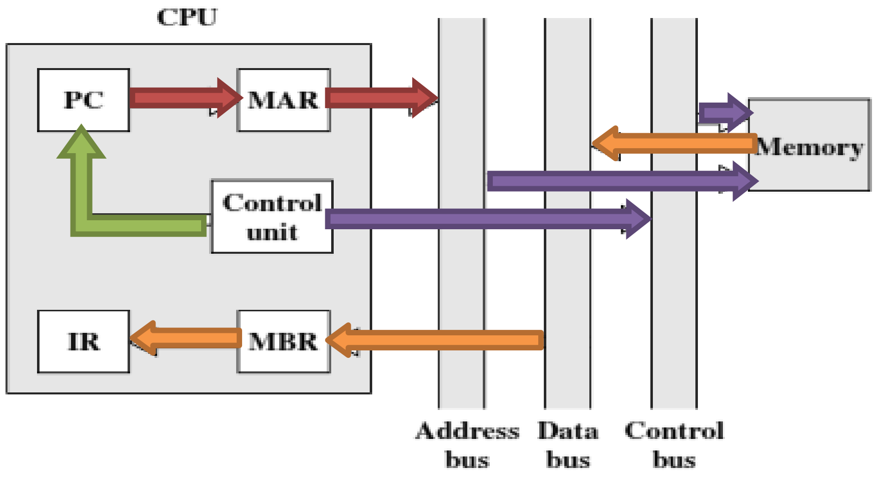
State diagram of an Instruction cycle

Fetch cycle

In the *fetch cycle,* instruction is read from memory.

1. Program counter (PC) holds the address of the next instruction to be fetched. Its address is given to MAR and placed on the address bus
2. The control unit issues the memory read signal and result is placed on data bus. This is copied into MBR, then moved to IR.
3. PC is incremented by 1, to fetch next instruction.

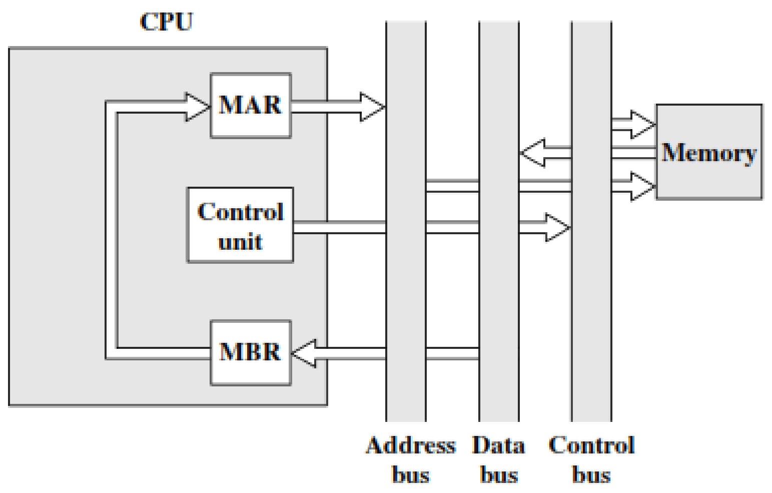
Fetch cycle



MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

Source: William Stallings, Computer Organization and Architecture,9th Edn

Data Flow



Source: William Stallings, Computer Organization and Architecture,9th Edn

Data Flow (Decode/Execute cycle)

- The *execute cycle* depends on machine instructions present in IR.
- involves data transfer between registers, memory or I/O

Instruction pipelining

- pipelining of Instructions is to speed up the execution time of processor
- Pipelining involves breaking up the instruction cycle into a number of separate stages that occur in sequence (e.g., fetch instruction, decode instruction, determine operand addresses, fetch operands etc).
- **Analogy**: manufacturing assembly lines;
  - By laying the production process out in an assembly line, products at various stages can be worked on simultaneously.
  - This process is also referred to as *pipelining,* because, as in a pipeline, new inputs are accepted at one end before previously accepted inputs appear as outputs at the other end.

Pipelining strategy

- Example: subdividing instruction

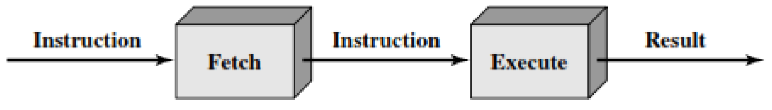
  processing into two stages:

  fetch instruction and execute instruction.

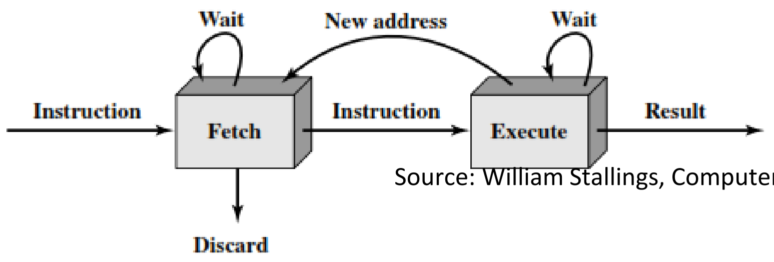
- During the execution of an instruction, there a times when main memory is not being accessed. This time could be used to fetch the next instruction in parallel with the execution of the current one.

Pipelining strategy

- The **first stage** fetches an instruction and buffers it. When the second stage is free, the first stage passes the buffered instruction (to be executed).
- While the **second stage** is executing the instruction, the first stage takes advantage of any unused memory cycles to fetch and buffer the next instruction. This is called *instruction prefetch or fetch overlap.*



(a) Simplified view



Source: William Stallings, Computer Organization and Architecture,9th Edn

(b) Expanded view

Pipelining strategy (cont…)

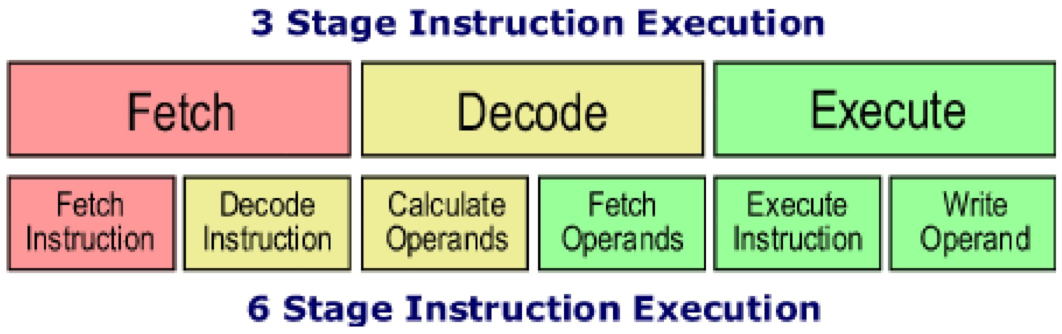
- Note: pipelining requires registers to store data between stages.
- This process will speed up instruction execution.
- If the fetch and execute stages were of equal duration, the instruction cycle time would be halved.

Pipeline - decomposition

- Pipelining can severely cut the time taken but the instruction execution process do not necessarily take an equal amount of time.
  - E.g. Time taken to 'execute' generally longer than 'fetch‘ and this makes it much harder to synchronise the various stages of the different instructions.
  - Also, some instructions may be dependent on the results of other earlier instructions. This can arise when data produced earlier needs to be used,
  - when a conditional branch based on a previous outcome is used.

Pipeline - decomposition

- Solution: breaking the instruction execution cycle into stages that are more likely to be of an equal duration. For example, the diagram below shows how the cycle can be broken down into six stages rather than three:



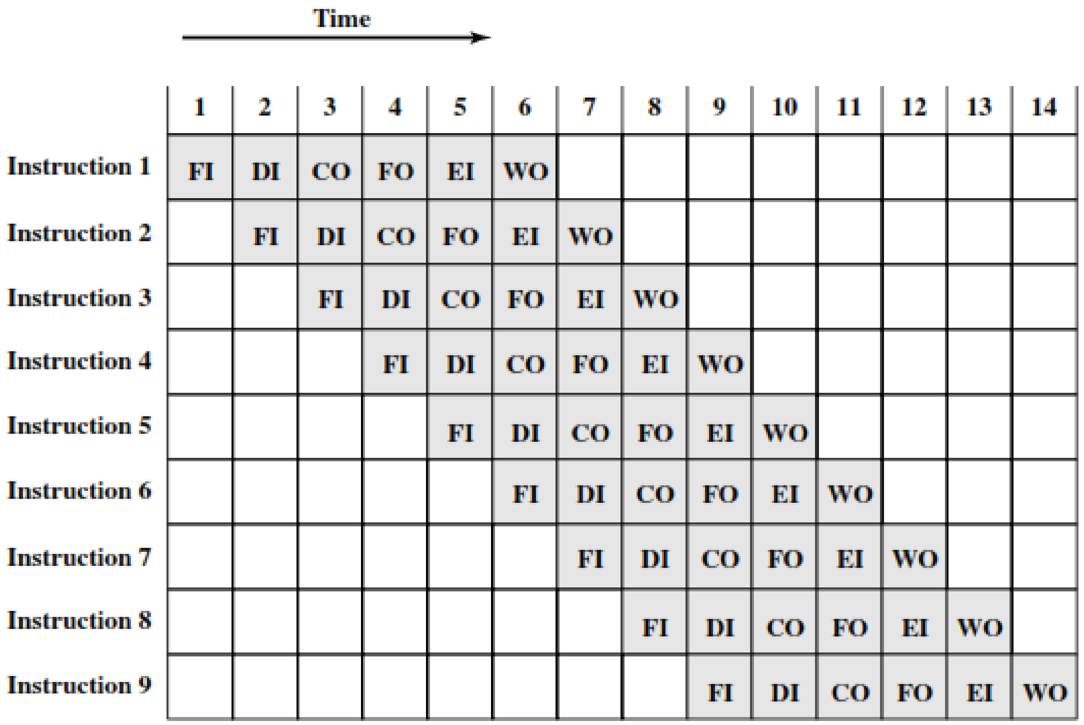
Source: William Stallings, Computer Organization and Architecture,9th Edn

Pipelining - decomposition

- Pipeline contain several stages. Consider the following decomposition of the instruction processing.
  - Fetch instruction (FI): Read the next expected instruction into a buffer.
  - Decode instruction (DI): Determine the opcode and the operand specifiers.
  - Calculate operands (CO): Calculate the effective address of each source operand. This may involve displacement, register indirect, indirect, or other forms of address calculation.
  - Fetch operands (FO): Fetch each operand from memory. Operands in registers need not be fetched.
  - Execute instruction (EI): Perform the indicated operation and store the result, if any, in the specified destination operand location.
  - Write operand (WO): Store the result in memory.

Pipelining – decomposition (cont…)

- The below figure shows a six-stage pipeline reduce the execution time for 9 instructions from 54 time units to 14 time units.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

Figure 12.10 Timing Diagram for Instruction Pipeline Operation

Source: William Stallings, Computer Organization and Architecture,9th Edn

Pipelining – decomposition (cont…)

Assumptions (see figure 12.10)

- All instruction are equal duration
- Each instruction goes through all six stages of the pipeline (This will not always be the case)
- All of the stages can be performed in parallel.
- There are no memory conflicts. For example, the FI, FO, and WO stages involve a memory access. The diagram implies that all these accesses can occur simultaneously. Most memory systems will not permit that. However, the desired value may be in cache, or the FO or WO stage may be null. Thus, much of the time, memory conflicts will not slow down the pipeline.

Pipelining: Advantages

- **The cycle time of the processor is reduced by increasing the instruction throughput**. Pipelining doesn't reduce the time it takes to complete an instruction; instead it increases the number of instructions that can be processed simultaneously ("at once") and reduces the delay between completed instructions (called 'throughput').

  The more pipeline stages a processor has, the more instructions it can process "at once" and the less of a delay there is between completed instructions.
- If pipelining is used, the CPU Arithmetic logic unit can be designed faster

Pipelining: Disadvantages

- The design of a non-pipelined processor simpler and cheaper to manufacture, non-pipelined processor executes only a single instruction at a time.
- In pipelined processor, insertion of flip flops between modules increases the instruction latency compared to a non-pipelined processor.

Pipelining: Disadvantages

- Many designs include pipelines as long as 7, 10, 20, 31 and even more stages; a disadvantage of a long pipeline is when a program *branches*, the entire pipeline must be flushed (cleared). The higher throughput of pipelines falls short when the executed code contains many branches: the processor cannot know in advance where to read the next instruction, and must wait for the branch instruction to finish, leaving the pipeline behind it empty.

  *This can be solve by predicting whether the conditional branch instruction will branch based on previous activity. After the branch is resolved, the next instruction has to travel all the way through the pipeline before its result becomes available and the processor resumes "working" again.
- When a programmer (or compiler) writes assembly code, they generally assume that each instruction is executed before the next instruction is being executed. When this assumption is not validated by pipelining it causes a program to behave incorrectly, thus is known as a **'hazard'**.
  *Various techniques for resolving hazards or working around such as forwarding and delaying (by inserting a stall or a wasted clock cycle) exist.

CHAPTER 7 ENDS!