Computer Architecture

Keivan Navi

Cal Poly Pomona University

knavi@cpp.edu

Office hours: Tu/Th 5:25 Pm to 6:55 Pm

Office: 8-49

Computer Architecture

Pipeline

Pipeline

- Instruction pipelining is a technique that improves processor performance by breaking down instructions into stages and executing them simultaneously.
- In pipelining, different parts of the processor work on different parts of instructions at the same time. This is similar to an assembly line, where each worker completes a part of the task.
- Pipelining increases the throughput of the system by allowing processes to overlap. ***The efficiency of the pipeline can be improved by dividing the instruction cycle into segments of equal duration.***
- To make processors even faster, various methods of optimizing pipelines have been devised. One method is branch prediction, where the processor guesses which path to take. If the guess is wrong, the pipeline must be restarted with the correct instruction.

Examples of Instruction pipelinig

- An example of two stage pipeline follows:
- **Fetching**: Reading instructions from memory **Executing**: Carrying out instruction
- Example of 4 stages pipeline:
- **Fetching**: Reading instructions from memory
- **Decoding**: Finding out what is this instruction
- **Executing**: Carrying out instructions
- **Writing Back**: Storing the final result into the Register Bank (Register File), if necessary.
- Example of 5 stages pipeline:
- **Fetching**: Reading instructions from memory
- **Decoding**: Finding out what is this instruction
- **Executing**: Carrying out instructions
- **Memory Access**: Accessing the memory, if necessary
- **Writing Back**: Storing the final result into the Register

Description of the 7-stage MIPS pipeline

- **Fetching** (Reading instructions from memory):
- Because fetching instruction from memory take a lot of time, we divide this part to two parts:
  - **Fetch1**
  - **Fetc**h2
- **Decoding**: Finding out what is this instruction
- **Executing**: Carrying out instructions
- **Memory Access**: (Accessing the memory, if necessary):
- Because Accessing memory take a lot of time, we divide this part to two parts:
  - **MemoryAccess1**
  - **MemoryAccess2**
- **Writing Back**: Storing the final result into the Register

Examples: Pentium, Pentium 4, Core i5, Core i7, Core i9, … Pipeline Stages

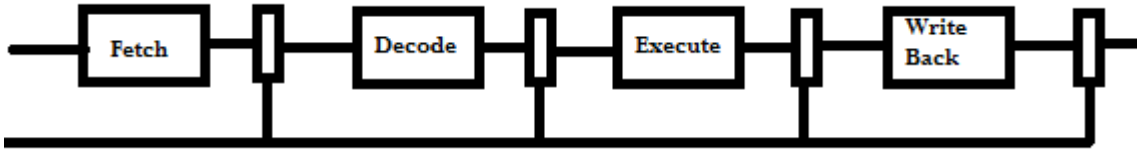
- **Pentium**
- The basic CPU pipeline has five stages: instruction fetch, decode instructions, fetch operands, execute, and store to cache. The Pentium also has a floating point pipeline with eight stages to speed up the floating point unit
- **Pentium Pro**
- The Pentium Pro has a 14-stage pipeline.
- **Pentium 4**
- The Pentium 4 has a 20-stage pipeline, which Intel calls Hyper Pipelined Technology. The deeper the pipeline, the more stages an instruction must go through before reaching the end of the pipeline.
- **Core i5**: A typical Intel Core i5 processor generally has a 5-stage pipeline. This means that an instruction goes through five distinct steps during processing: instruction fetch, instruction decode, execute, memory access, and write back.
- **Core i7**: The Intel Core i7 has 14 pipeline stages. The pipeline depth of the Core i7 allows for out-of-order execution of micro-operations (µops) to fill the pipeline.
- Core i9: ?

The Physical limit of pipe line stages

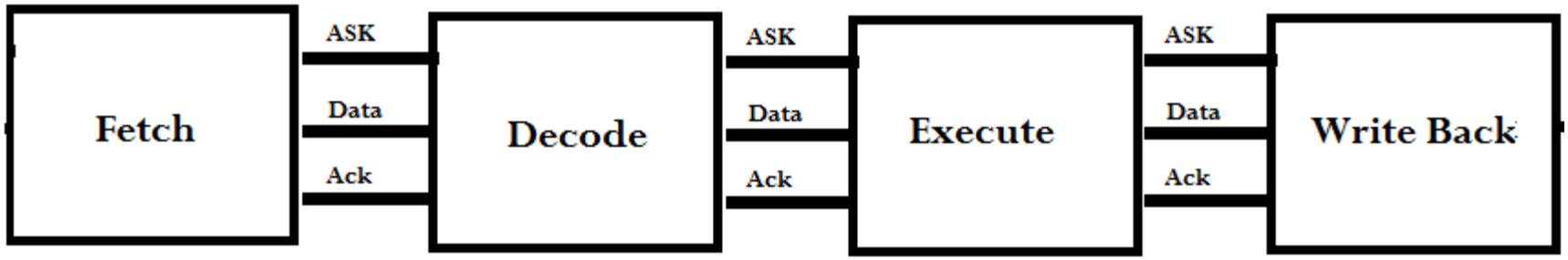
- The physical limit of a pipeline's number of stages is primarily constrained by factors like pipeline hazards, sequencing overhead, and cost; while technically there's no absolute limit, very long pipelines become impractical due to increased complexity in managing dependencies between stages, potentially negating the performance benefits of adding more stages; most modern processors typically have pipelines with 5 to 20 stages depending on the design and application.
- Although, all of them can be some factors of the physical limit of the number of the stages but this one is the major one:
- ***The Delay of the stage must be much more than the delay of the Register (it will explained later).***

Synchronous Pipeline versus Asynchronous Pipeline

- Synchronous Pipeline:



- Between each to stages one register must be added, in order to synchronize the operation.
- The delay of each stage must be at least ten time greater the delay of the register. Why?
- Asynchronous Pipeline

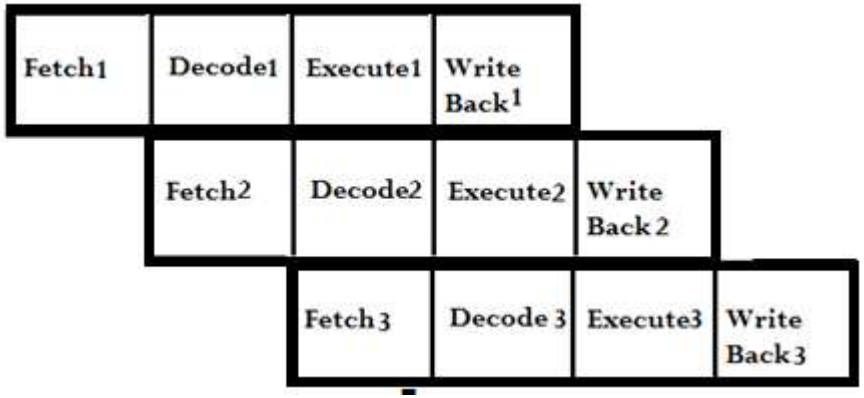


Pipeline Versus Non-Pipeline

- Non Pipeline (3 Instructions)

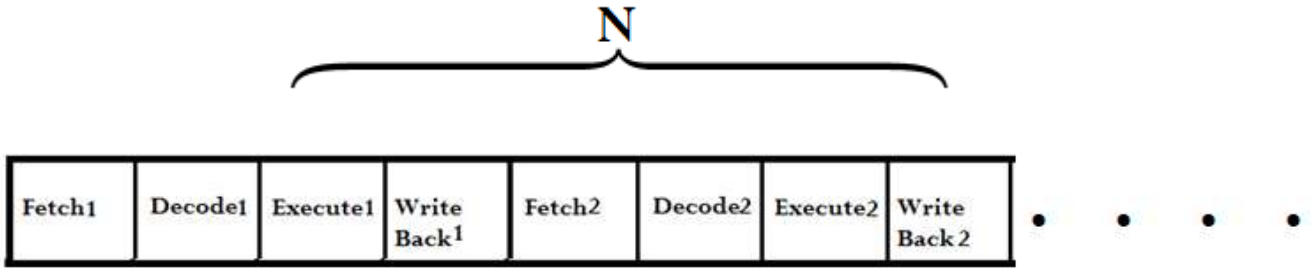
Fetch1	Decode1	Execute1	Write Back1	Fetch2	Decode2	Execute2	Write Back 2	Fetch 3	Decode 3	Execute3	Write Back 3

- Pipeline (3 Instructions)

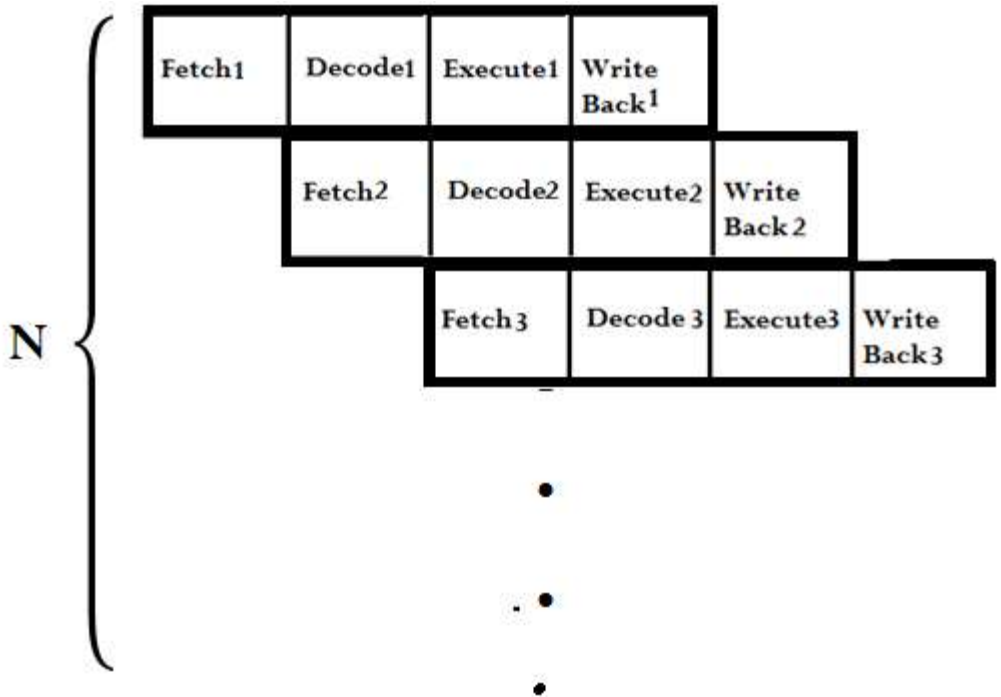


Pipeline Versus Non-Pipe (continued)

- Non Pipeline N Instructions:



- Pipeline N Instructions:



Speed Up of Pipeline

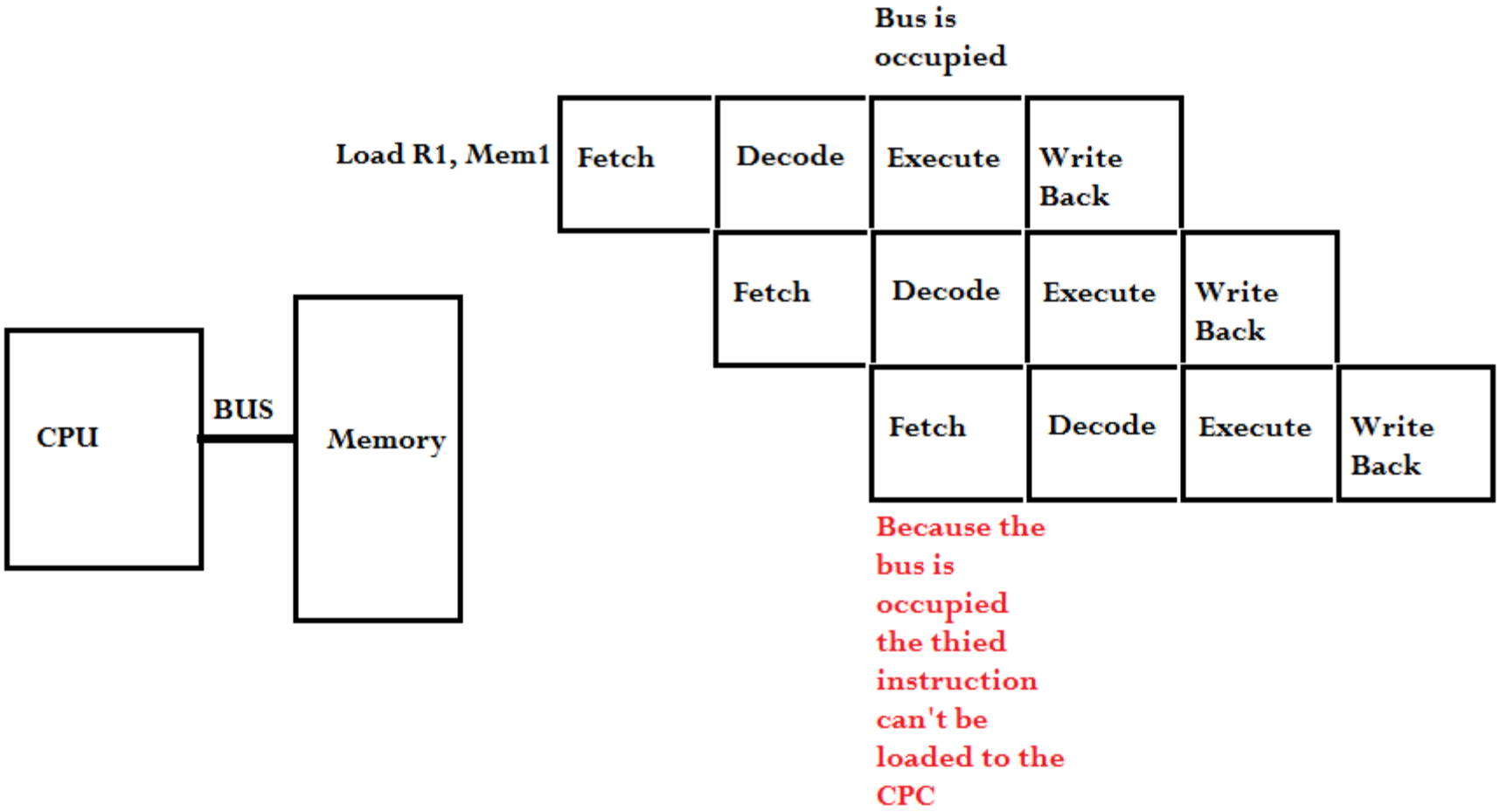
- This table illustrates the delay versus the number of instruction for pipeline CPU and non-pipeline CPU:

Number of instructions	Delay without pipelining	Delay with pipelinig
1	kt	kt
2	2kt	kt+1t
3	3kt	kt+2t
4	4kt	kt+3t
.	.	.
.	.	.
.	.	.
.	.	.
N	Nkt	kt+(N-1)t

Speed Up (continued)

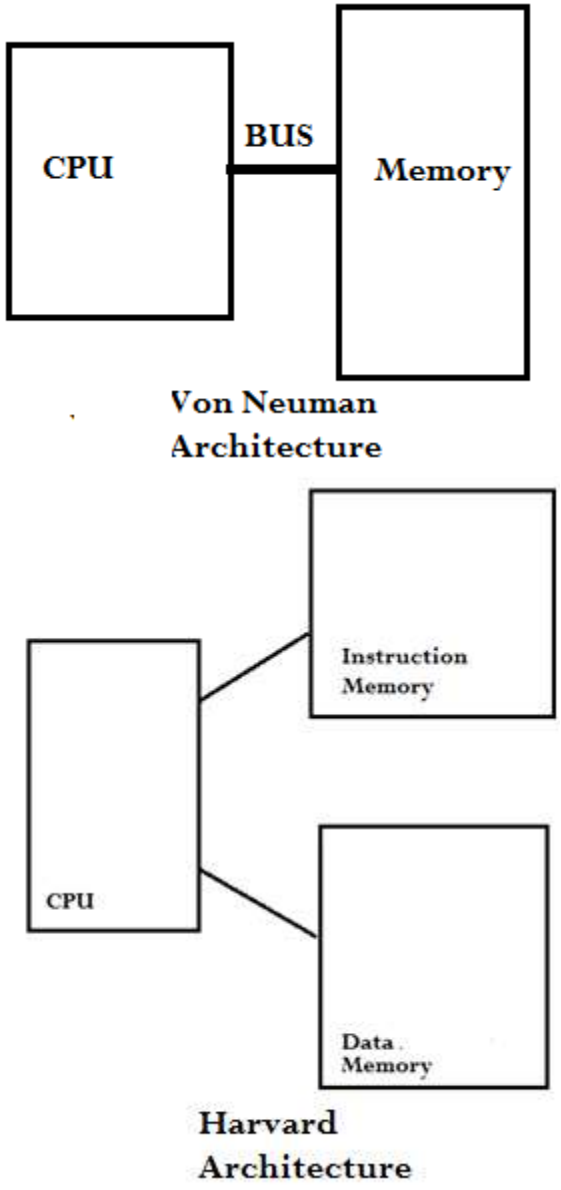
- Speed up = $\frac{Delay\ without\ pipeline}{Delay\ with\ pipeline}$
- Speed up= $\frac{Nkt}{kt+(N-1)t} = \frac{Nk}{k+(N-1)}$
- $\underset{N\to\infty}{Speed\ up} Max$ = k
- We can never reach the maximum Speed up in a real program, Because of the instruction dependency. Some Examples of Instruction Dependency follows:
- What is the drawback in a 4 stage pipeline if the first instruction is “LOAD”?
- The third instruction can’t be loaded to the CPU because the bus is already occupied for bringing the data of the first instruction from the memory to the CPU. In the nest page it is illustrated.

Example: Conflict when the first instruction is load



How can we solve it?

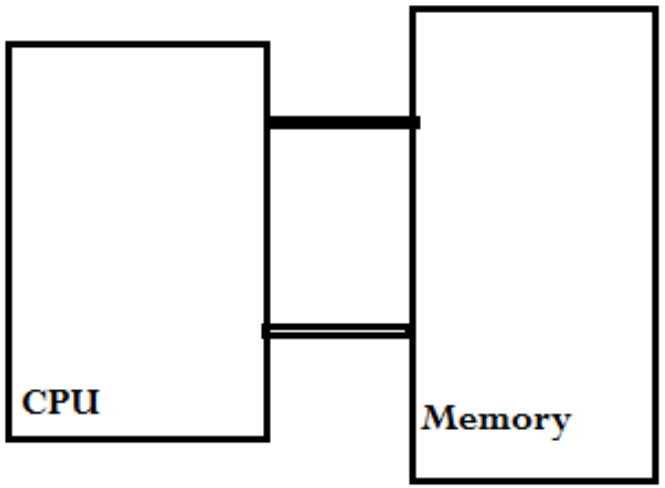
- Harvard Architecture: Harvard architecture is a computer architecture that separates instructions and data storage. The processor connects to two independent memory banks, each with its own set of buses. One bank stores program instructions, and the other stores data. Harvard architecture can lead to faster instruction execution and better performance in some applications. Harvard architecture is often compared with von Neumann architecture, where instructions and data share the same memory and pathways.
- The problem is that it not flexible. If in some cases we need more Data Memory or Instruction Memory, we can't replace any part of these two memory in the favor of the other one.



Another Solution

- Another Solution which is more flexible is to use two buses with the same shared memory.

- A CPU typically communicates with memory using a single "system bus," which consists of multiple lines including a data bus (for transferring data), an address bus (specifying memory locations), and a control bus (managing data transfer operations); however, in some advanced architectures, a CPU might utilize two separate buses for accessing memory, which is better than the Harvard Architecture from flexibility point of view.



Two Buses with the same shared memory

Example: Resource Dependency

- In some cases the next instruction needs the data of the previous instruction.
- Data dependency is a relationship between instructions or statements in a program where the output of one is dependent on the input of another. It can occur when a statement writes data that is later read by another statement.
- Data dependency can lead to issues when executing instructions concurrently, so it must be managed to ensure the program runs correctly. For example, in serial programming, the program executes statements in the order they appear in the code. To manage data dependency in this case, you write the code statements in the correct order. In parallel programming, where some parts of the code are executed simultaneously, managing data dependency is more complicated. Here is an example of Data Dependency:
- Add R1, R2, R3
- Sub R4, R1, R7
- In this case the second instruction needs the data of R1.