Instruction Pipelining

Pushpendra Kumar Pateriya

Performance

- It is the time taken by the system to execute a program.
- Performance affected by following parameters:
  - Clock speed
  - Type and number of instructions available
  - Average time required to execute an instruction
  - Memory access time
  - Power dissipation in the system
  - Number of I/O devices and types of I/O devices connected
  - The data transfer capacity of the bus

Important Fact

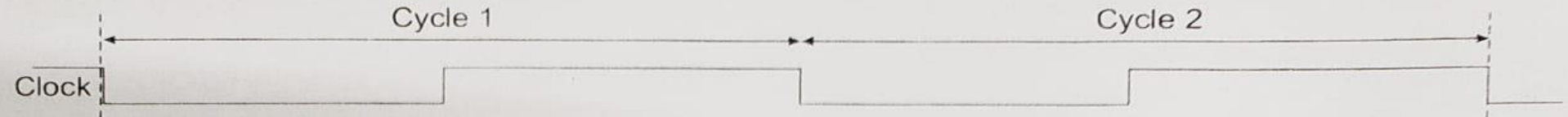
- CPI: Cycles per instruction
- IPC: Instructions per cycle
- IPC = 1/CPI
- CPU clock cycles = Number of instructions X Average clock cycles per instruction
- CPU clock cycles $= \sum_{i=1}^{n} N_i X CPI_i$
- CPU execution time = CPU clock cycles X Clock cycle time
- CPU execution time = CPU clock cycles / Clock rate
- CPU execution time = Instruction Count X CPI X Clock Cycle Time
- CPU execution time = $\frac{Instructions}{Program} X \frac{Clock\ Cycle}{Instruction} X \frac{Seconds}{Clock\ Cycle}$
- CPU execution time = Seconds / Program
- Speedup = (Earlier execution time) / (Current execution time)

Instruction Processing

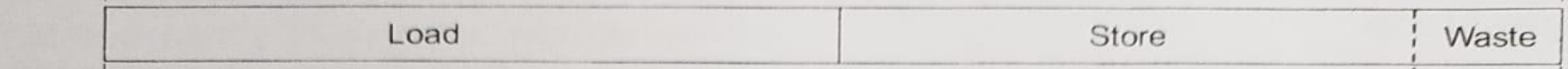
Single Cycle	Multiple Cycle	Pipeline
Cycle time long enough for the longest instruction	Cycle time long enough for the longest stage	Cycle time long enough for the longest stage
Entire cycle acts as a single stage	Shorter stages waste time	Shorter stages waste time
Shorter instruction waste time	Shorter instructions can take fewer cycles	No additional benefit from shorter instructions
No overlap	No overlap	Overlap instruction execution

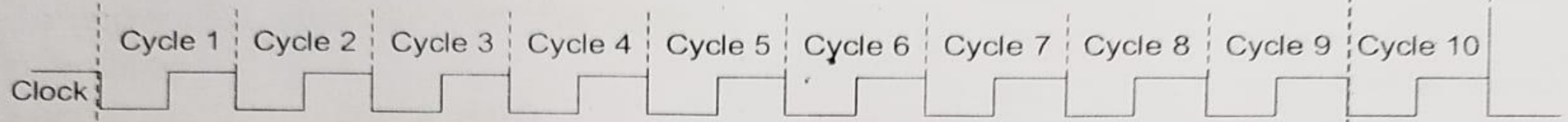
5 Stage Instruction Cycle

- Instruction Fetch
- Decode (Interpret) Instruction
- Fetch Operands
- Perform Operation
- Store Result

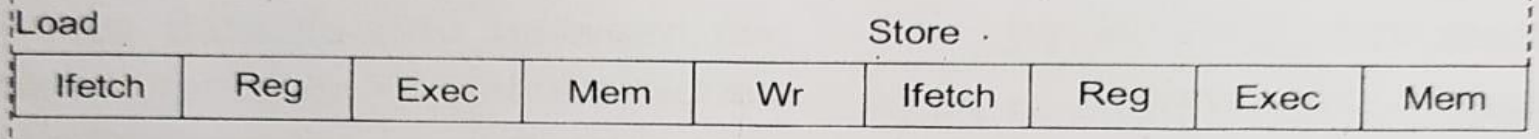


Single Cycle Implementation:

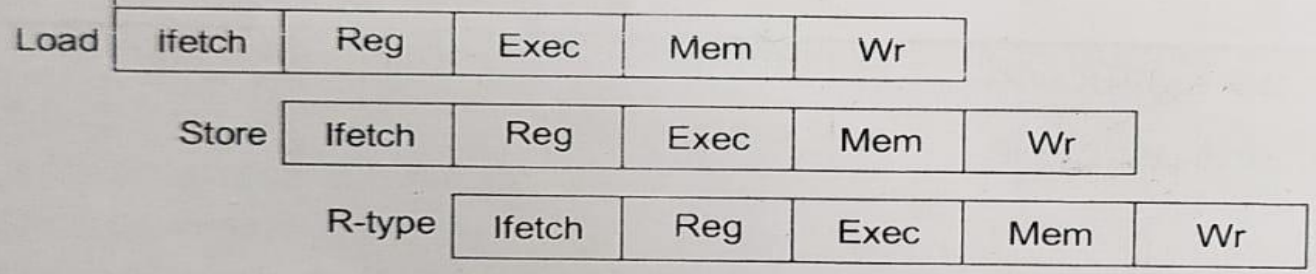




Multiple Cycle Implementation:



Pipeline Implementation:



Where **Ifetch (Instruction Fetch):** Fetch the instruction from the Instruction Memory
Reg/Dec: Registers Fetch and Instruction Decode
Exec: Calculate the memory address
Mem: Read the data from the Data Memory
Wr: Write the data back to the register file

Example: Suppose we execute 100 instructions:

Single Cycle Machine: 45 ns/cycle × 1 CPI x 100 inst = 4500 ns

Multi-cycle Machine: 10 ns/cycle × 4.2 CPI (due to inst mix) × 100 inst = 4200 ns

Ideal 5 stages pipelined machine: 10 ns/cycle x (1 CPI × 100 inst + 4 cycle drain) = 1040 ns

3.3 Differences between Datapaths

	Single Cycle	Multiple Cycle	Pipeline
Instructions Subdivided?	No	Yes, into arbitrary number of steps	Yes, into one step per pipeline stage
Clock Cycle Time	Long (long enough for the slowest instruction)	Short (long enough for the slowest instruction step)	Short (long enough for the slowest pipeline stage)
CPI	1 clock cycle per instruction (by definition)	Variable number of clock cycles per instruction	Fixed number of clock cycles per instruction, one for each pipeline stage (under ideal conditions)
Number of Instructions Executing at the Same Time	1	1	As many instructions as pipeline stages (under ideal conditions)
Control Unit	Generates signals for entire instruction	Generates signals for instruction's current step, and keeps track of the current step	Generates signals for entire instruction; these signals are propagated from one pipeline stage to another via the pipeline registers
Duplicate Hardware?	Yes, since we can use a functional unit (FU) for at most one subtask per instruction	No, since the instruction generally is broken into single-FU steps	Yes, so that there are no restrictions on which instructions can be in the pipeline simultaneously
Extra Registers?	No	Yes, to hold the results of one step for use in the next step	Yes, to provide the results of one pipeline stage to the next pipeline stage
Performance	Baseline	Slightly slower to moderately faster than single cycle, the latter when the instructions steps are well balanced and a significant fraction of the instructions take less than the maximum number of clock cycles	Moderately faster to significantly faster than single cycle, the latter if the pipeline stages are well balanced and the pipeline handles hazards well

Pipeline Design and Issue

CPU with 5-stage Pipeline Hardware

Let us consider the following decomposition of the instruction execution into five stages:

1. **Fetch Instruction (IF):** Read the next expected instruction into a buffer.
2. **Instruction Decoding (ID):**
   - Decode Instruction: Determine the opcode and the operand specifiers.
   - Calculate Operand: Calculate the effective address of each source operand.
   - Fetch Operands: Fetch each operand from memory.
3. **Execute Instruction (EX):** Perform the indicated operation.
4. **Memory access/branch completion cycle (MEM):** Access the memory
5. **Write Operand (WO):** Store the result.

Problems in Instruction Pipelining

- Timing Variations
- Data Hazards
- Branching
- Interrupts

Pipeline Hazards

- Structural Hazard
  - Different instructions trying to access the same piece of H/W in the same segment of pipeline.
- Data Hazard
  - Instruction depends on the result of a prior instruction
    - RAW (True Data Dependency/Flow Dependency)
    - WAR (Antidependency)
    - WAW (Output Dependency)
- Control Hazard

Berstein's Conditions

For RAW : $O(i)) \cap I(j) = \phi$

For WAR : $I(i)\ ) \cap O(j) = \phi$

For WAW : $O(i)\ ) \cap O(j) = \phi$

Example-3.1 For the following code sequence check the existence of Data Hazards.

1. ADD R3, R2, R1 ; R3 = R2 + R1
2. SUB R5, R1, 1 ; R5 = R1 – 1

Solution:

$O(1) = \{R3\}, O(2) = \{R5\}, I(1) = \{R1, R2\}$ and $I(2) = \{R1\}$

The conditions:

$\{R3\} \cap \{R1\} = \phi$ (RAW)

$\{R2, R1\} \cap \{R5\} = \phi$ (WAR)

$\{R3\} \cap \{R5\} = \phi$ (WAW)

All conditions are satisfied hence there are no data hazards in the code.

Handling Data Hazards

- Forwarding: (Add special circuitry, which one forwards the desired value to the pipeline segment)
- Code Re-Ordering
- Stall Insertion: (no-op instruction)

Control Hazard

- Can result from branch instructions
- Types of control hazards:
  - Jump to the branch target address if the instruction succeeds
  - Execute the instruction after the branch instruction if the branch fails

3.4.4 Performance Analysis with Pipeline

$$\text{Cycle time } \tau = \text{Max}(\tau_i) + d = \tau_m + d;\ 1 \le i \le k$$

where τ_m = Maximum stage delay (delay through stage which experience the largest delay)

k = Number of stages in the instruction pipeline.

d = Time delay of a latch, needed to advance signals and data from one stage to the next.

In general, the time delay d is equivalent to a clock pulse and $\tau_m >> d$. The total time required τ_k to execute all n instructions is

3.7 Speedup

- Ideal $CPI_{pipeline} = 1$
- $CPI_{pipeline}$ = Ideal $CPI_{pipeline}$ + Pipelined stall cycles per instruction
  = 1 + Pipelined stall cycles per instruction
- $$\text{Speedup} = \frac{\text{Average\_instruction\_time}_{unpipelined}}{\text{Average\_instruction\_time}_{pipelined}}$$

$$= \frac{CPI_{unpipelined}}{CPI_{pipelined}} \times \frac{\text{Clock\_cycle}_{unpipelined}}{\text{Clock\_cycle}_{pipelined}}$$

$$= \frac{CPI_{unpipelined}}{1 + \text{Pipeline stall cycles per instruction}} \times \frac{\text{Clock\_cycle}_{unpipelined}}{\text{Clock\_cycle}_{pipelined}}$$

$$= \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall cycles per instruction}}$$

- Computation of speedup with branch penalty:

$$\text{Pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} = \frac{\text{Pipeline depth}}{1 + \text{Branch Frequency} \times \text{Branch penalty}}$$

- If **"A is n times faster than B"**:

$$n = \frac{\text{Execution\_time}_B}{\text{Execution\_time}_A} = \frac{1/\text{Performance}_B}{1/\text{Performance}_A} = \frac{\text{Performance}_A}{\text{Performance}_B}$$

- If **"A is n% faster than B"**:

$$n = \left(\frac{\text{Performance}_A - \text{Performance}_B}{\text{Execution}_B}\right) \times 100 = \left(\frac{\text{Execution\_Time}_B - \text{Execution\_Time}_A}{\text{Execution\_Time}_A}\right) \times 100$$

Example-3.2 If machine A runs a program in 10 seconds and machine B runs the same program in 15 seconds, which of the following statements is true?

(a) A is 50% faster than B

(b) A is 33% faster than B

Solution:

Machine A is n% faster than machine B can be expressed as:

$$\frac{\text{Execution\_Time}_B}{\text{Execution\_Time}_A} = 1 + \frac{n}{100}$$

$$\Rightarrow \quad n = \left(\frac{\text{Execution\_Time}_B - \text{Execution\_Time}_A}{\text{Execution\_Time}_A} \right) \times 100 = \frac{15 - 10}{10} \times 100 = 50$$

Therefore A is 50% faster than B.

Example - 3.3 CPU clock frequency is 1 MHz (clock cycle time is 10^{-6} sec.) Program takes 4.5 million cycles to execute. What is the CPU time?

Solution:

CPU time = $4{,}500{,}000 \times 10^{-6} = 4.5$ seconds

Example-3.4 CPU clock frequency is 500 MHz (clock cycle time is 10^{-6} sec.) Program takes 45 million cycles to execute. What is the CPU time?

Solution:

CPU time = 4,500,000 × (1/500,000,000) = 0.09 seconds

Example-3.5 A benchmark has 100 instructions with the following information:

(a) 25 instructions are loads/stores (each takes 2 cycles)

(b) 50 instructions are adds (each takes 1 cycles)

(c) 25 instructions are square root (each takes 100 cycles)

What is the CPI?

Solution:

CPI = (25 × 2) + (50 × 1) + (25 × 100) / 100 = 2600/100 = 26.0

Calculating average CPI for the following table.

OP	Freq	CPI_i
ALU	50%	1
Load	20%	2
Store	10%	2
Branch	20%	2
	100%	

OP	Freq	CPI_i	$CPI_i \times F_i$	(% time)
ALU	50%	1	0.5	(33%)
Load	20%	2	0.4	(27%)
Store	10%	2	0.2	(13%)
Branch	20%	2	0.4	(27%)
	100%		1.5	

Example-3.7 Program runs for 100 seconds on a uniprocessor. 10% of the program can be parallelized on a multiprocessor (F = 0.1). Assume a multiprocessor with 10 processors (n = 10). Compute the speedup.

Solution:

$$\text{Speedup} = \frac{1}{(1-0.1)+\frac{0.1}{10}} = \frac{1}{0.01+0.9} = \frac{1}{0.91} = 1.1$$

Example-3.8 Floating point instructions improved to run 2X; but only 10% of actual instructions are FP. Assume a multiprocessor with 10 processors (n = 10). Compute overall speedup.

Solution:

$$\text{Execution\_Time}_{new} = \text{Execution\_Time}_{old} \times \left(0.9 + \frac{0.1}{2}\right) = 0.95 \times \text{Execution\_Time}_{old}$$

$$\text{Speedup}_{overall} = \frac{1}{0.95} = 1.053$$

Consider a 4 stage pipeline processor. The number of cycles needed by the four instructions I1, I2, I3, I4 in stages S1, S2, S3, S4 is shown below:

	S1	S2	S3	S4
I1	2	1	1	1
I2	1	3	2	2
I3	2	1	1	3
I4	1	2	2	2

What is the number of cycles needed to execute the following loop?

For (i=1 to 2) {I1; I2; I3; I4;}

(A) 16

(B) 23

(C) 28

(D) 30

A ALWAYS
S SEEK
K KNOWLEDGE

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
I1	S1	S1	S2	S3	S4																			
I2			S1	S2	S2	S2	S3	S3	S4	S4														
I3				S1	S1	X	S2	X	S3	X	S4	S4	S4											
I4						S1	X	S2	S2	S3	S3	X	X	S4	S4									
I1							S1	S1	X	S2	X	S3	X	X	X	S4								
I2									S1	X	S2	S2	S2	S3	S3	X	S4	S4						
I3										S1	S1	X	X	S2	X	S3	X	X	S4	S4	S4			
I4												S1	X	X	S2	S2	S3	S3	X	X	X	S4	S4	

i=1

i=2

	S1	S2	S3	S4
I1	2	1	1	1
I2	1	3	2	2
I3	2	1	1	3
I4	1	2	2	2