

ASSIGNMENT

Course Code	CSC208A
Course Name	Computer Organization and Architecture
Programme	B.Tech
Department	CSE
Faculty	FET

Name of the Student	Satyajit Ghana
Reg. No	17ETCS002159
Semester/Year	03/2018
Course Leader/s	Naveeta Rani

Declaration Sheet			
Student Name	Satyajit Ghana		
Reg. No	17ETCS002159		
Programme	B.Tech	Semester/Year	03/2018
Course Code	CSC204A		
Course Title	Computer Organization and Architecture		
Course Date		to	
Course Leader	Naveeta Rani		
<p>Declaration</p> <p>The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.</p>			
Signature of the Student		Date	
Submission date stamp (by Examination & Assessment Section)			
Signature of the Course Leader and date		Signature of the Reviewer and date	

Declaration Sheet	ii
Contents	iii
List of Tables	iv
List of Figures	v
Question No. 1	6
A 1.1 Introduction to superscalar processors:	6
A 1.2 Discussion on applications of superscalar processors:	6
A 1.3 Conclusion:	7
Question No. 2	8
B 1.0. Data and Assumptions:	8
B 1.1. Explanation of the pipelined mode of execution of the instructions using a suitable diagram depicting various instruction execution phases, for each instruction:.....	10
B 1.2. Computation of the number of clock cycles with stage delay as 0ns for 5 stage pipeline processor:	12
Question No. 3	13
B 2.1. Introduction to split stages:	13
B 2.2 Compute the number of clock cycles with stage delay as 0ns including pipeline bubbles for 7 stage pipeline processor	13
B 2.3 Analyze the performance of 5 stage and 7 stage pipelined processor	14
B 2.4 Based on the analysis, recommend the best design for high performance computing applications:	16
B 2.5 Conclusion:	16

Table No.	Title of the table	Pg.No.
------------------	---------------------------	---------------

Figure No.	Title of the figure	Pg.No.
Figure B1.1	MIPS Instructions	8
Figure B1.2	Assembled Instructions	8
Figure B1.3	Status of registers after execution	9
Figure B1.4	Single Cycle Data Path	11

Solution to Question No. 1 Part A:

A 1.1 Introduction to superscalar processors:

Scalar processors are pipelined processor that are designed to fetch and issue at most one instruction every machine cycle. Super scalar processors are those that are designed to fetch and issue multiple instructions every machine cycle. Superscalar processor allows the processor pipeline to simultaneously process multiple instructions in each pipeline stage. By being able to sustain the execution of multiple instructions in every machine cycle, the CPI can be significantly reduced. Of course, the complexity of each pipe stage can increase, leading to a potential increase of cycle time or the pipeline depth, which can in turn increase the CPI.

The key microarchitecture technique for reducing cycle time is pipelining. Pipelining effectively partitions the task of processing an instruction into multiple stages. The latency (in terms of signal propagation delay) of each pipe stage determines the machine cycle time. By employing pipelines, the latency of each pipe stage, and hence the cycle time, can be reduced. In recent years, aggressive pipelining has been the major technique used in achieving phenomenal increase of clock frequency of high-end microprocessors.

Traditional supercomputers are parallel processors that perform both scalar and vector computations. During scalar computation only one processor is used. During vector computation all N processors are used to perform operations on array data. One formulation of Amdahl's law state the efficiency E of the parallel machine

$$E = \frac{h + N \times (1 - h)}{N} = 1 - h \times \left(1 - \frac{1}{N}\right)$$

We now restate Amdahl's law that models the performance of a parallel processor:

$$S = \frac{1}{(1 - f) + (f/N)}$$

This model gives the performance or speedup of a parallel system over that of a nonparallel system. The machine parallelism is measured by N , the number of processors in the machine. The parameter f is the vectorizability of the program which reflects the program parallelism. The formulation of this model is influenced by traditional supercomputers that contain a scalar unit and a vector unit.

A 1.2 Discussion on applications of superscalar processors:

Some of common applications of superscalar processors are:

- Science and Engineering
 - Weather prediction
 - Evolution of galaxies
 - Oil reservoir simulation
- Commercial
 - Transaction processing
 - Database
 - Financial models

- Automobile crash test
- Drug development
- VLSI CAD
- Nuclear Bombs
- Involves true parallelism
- Multimedia/home
 - Speech recognition
 - Data compression/decompression
 - 3D-Graphics
- Involves throughput parallelism
- Flynn Taxonomy
 - SISD
 - Single Instruction Single Data
 - Standard sequential machines
 - SIMD
 - Single Instruction Multiple Data

The past few years have seen a revolution in high performance scientific computing applications. The sequencing of the human genome by the International Human Genome Sequencing Consortium and Celera, Inc. has opened exciting new frontiers in bioinformatics. Functional and structural characterization of genes and proteins hold the promise of understanding and fundamentally influencing biological processes. Analyzing biological sequences with a view to developing new drugs and cures for diseases and medical conditions requires innovative algorithms as well as large-scale computational power. Indeed, some of the newest parallel computing technologies are targeted specifically towards applications in bioinformatics.

Advances in computational physics and chemistry have focused on understanding processes ranging in scale from quantum phenomena to macromolecular structures. These have resulted in design of new materials, understanding of chemical pathways, and more efficient processes. Applications in astrophysics have explored the evolution of galaxies, thermonuclear processes, and the analysis of extremely large datasets from telescopes. Weather modeling, mineral prospecting, flood prediction, etc., rely heavily on parallel computers and have very significant impact on day-to-day life.

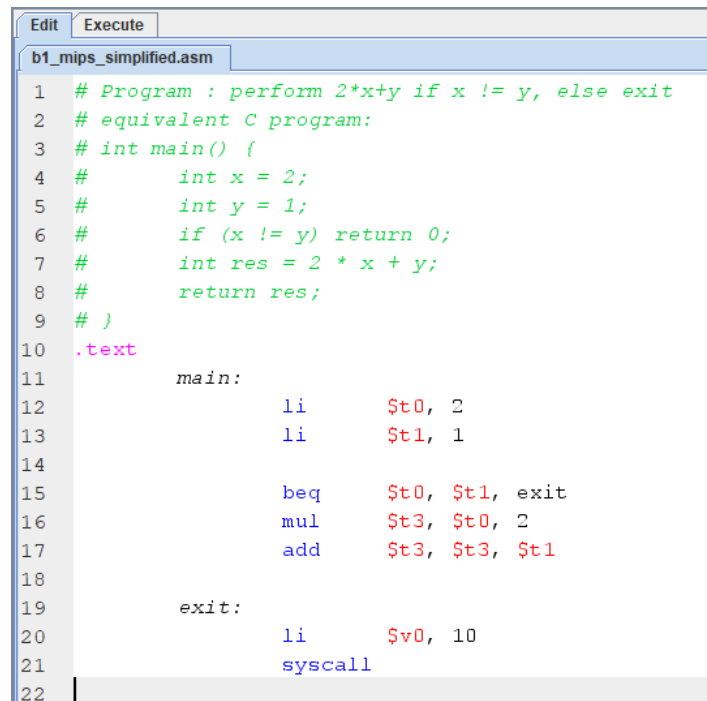
Bioinformatics and astrophysics also present some of the most challenging problems with respect to analyzing extremely large datasets. Protein and gene databases (such as PDB, SwissProt, and ENTREZ and NDB) along with Sky Survey datasets (such as the Sloan Digital Sky Surveys) represent some of the largest scientific datasets. Effectively analyzing these datasets requires tremendous computational power and holds the key to significant scientific discoveries.

A 1.3 Conclusion:

As computer systems become more pervasive and computation spreads over the network, parallel processing issues become engrained into a variety of applications. Superscalar processors finds its way into applications that we use on our everyday life, even a modern automobile consists of tens of processors communicating to perform complex tasks for optimizing handling and performance. The cost benefits of parallelism coupled with the performance requirements of applications present compelling arguments in favor of parallel computing.

Solution to Question No. 1 Part B:**B 1.0. Data and Assumptions:**

MIPS Assembly Instructions:



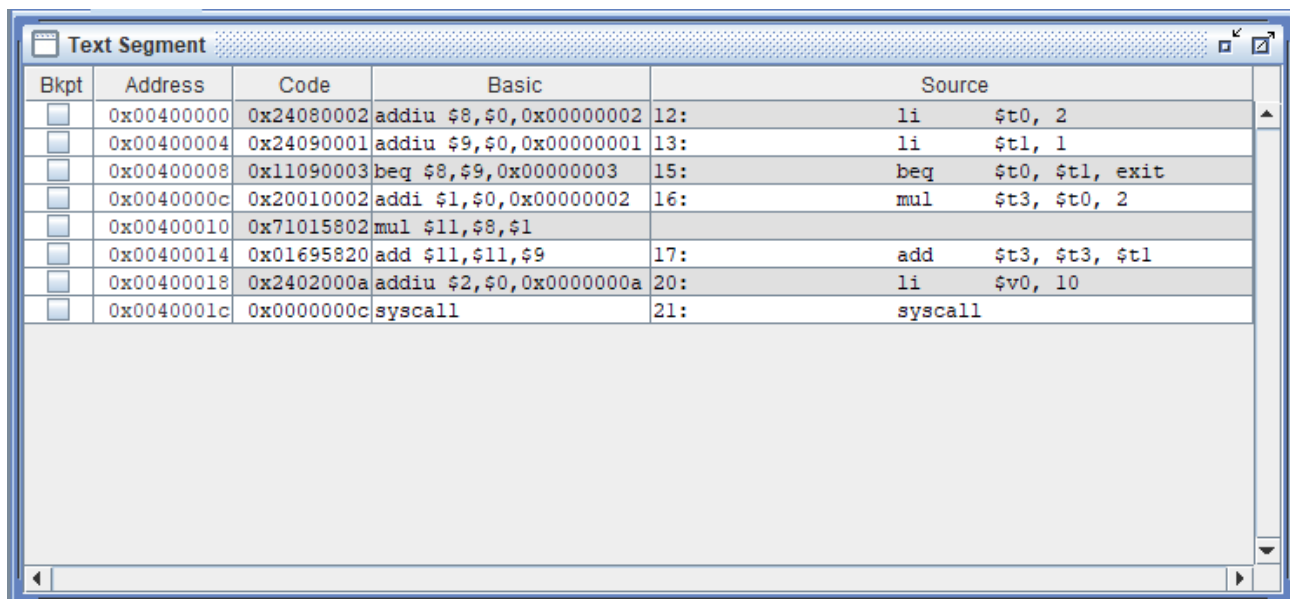
```

1  # Program : perform 2*x+y if x != y, else exit
2  # equivalent C program:
3  # int main() {
4  #     int x = 2;
5  #     int y = 1;
6  #     if (x != y) return 0;
7  #     int res = 2 * x + y;
8  #     return res;
9  # }
10 .text
11     main:
12         li      $t0, 2
13         li      $t1, 1
14
15         beq     $t0, $t1, exit
16         mul     $t3, $t0, 2
17         add     $t3, $t3, $t1
18
19     exit:
20         li      $v0, 10
21         syscall
22

```

Figure B1.1 MIPS Instructions

Assembling the source code:



Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24080002	addiu \$8,\$0,0x00000002	12: li \$t0, 2
<input type="checkbox"/>	0x00400004	0x24090001	addiu \$9,\$0,0x00000001	13: li \$t1, 1
<input type="checkbox"/>	0x00400008	0x11090003	beq \$8,\$9,0x00000003	15: beq \$t0, \$t1, exit
<input type="checkbox"/>	0x0040000c	0x20010002	addi \$1,\$0,0x00000002	16: mul \$t3, \$t0, 2
<input type="checkbox"/>	0x00400010	0x71015802	mul \$11,\$8,\$1	
<input type="checkbox"/>	0x00400014	0x01695820	add \$11,\$11,\$9	17: add \$t3, \$t3, \$t1
<input type="checkbox"/>	0x00400018	0x2402000a	addiu \$2,\$0,0x0000000a	20: li \$v0, 10
<input type="checkbox"/>	0x0040001c	0x0000000c	syscall	21: syscall

Figure B1.2 Assembled Instructions

Registers after execution of the instructions:

\$t0	8	0x00000002
\$t1	9	0x00000001
\$t2	10	0x00000000
\$t3	11	0x00000005
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000

Figure B1.3 Status of Registers after execution

Assumptions:

Assumptions:

1. Pipeline contains 5 stages, IF, ID, EXE, MEM, WB.
2. Each stage requires one clock cycle.
3. All memory references hit in cache

Time taken for each stage:

Stage	IF	ID	EXE	MEM	WB
Time Taken	100ps	200ps	400ps	200ps	150ps

The Instructions to be executed are:

```

1. main:
2.      li    $t0, 2          #INS1
3.      li    $t1, 1          #INS2
4.      beq   $t0, $t1, exit  #INS3
5.      mul   $t3, $t0, 2      #INS4
6.      add   $t3, $t3, $t1    #INS5
7. exit:
8.      li    $v0, 10
9.      syscall

```

Assumptions specific to the processor and the instructions:

1. The branch instruction is not taken.
2. The processor can do Operand Forwarding to prevent stalls and improve performance wherever necessary.
3. There is branch prediction wherever necessary.
4. ADD and MUL take the same number of CPU cycles for EXE stage.
5. R-Type Instruction does not have MEM stage hence NOP

6. Store Instruction does not have WB stage hence NOP
7. Branch Instruction does not have MEM and WB hence NOP

Abbreviations used:

1. IF: Instruction Fetch
2. ID: Instruction Decode
3. EXE: Execute
4. MEM: Memory
5. WB: Write Back
6. NOP: No Operation
7. STL: CPU Stall

B 1.1. Explanation of the pipelined mode of execution of the instructions using a suitable diagram depicting various instruction execution phases, for each instruction:

Pipelining is an implementation technique in which multiple instructions are overlapped in execution. It is the ability to overlap execution of different instructions at the same time, it exploits parallelism among instructions and is not visible to the programmer.

Under ideal conditions and with a large number of instructions, the speed-up from pipelining is approximately equal to the number of pipe stages; a five-stage pipeline is nearly five times faster.

Basic Idea:

- Split the instruction execution into multiple stages
 - Similar to multi-cycle implementation
- Start a new instruction in each clock cycle
- Different pipeline stages are completing different parts of different instructions in parallel

Ideal Pipeline

Assumptions:

- Perfectly balanced pipelining stages
 - Each stage takes exactly the same amount of time
- Instructions utilize the same hardware resource in every pipeline stage
 - Good utilization of hardware resource
- There are no stalls for dependencies
 - Instructions can always be executed in lock-step fashion.

Optimal Behaviour:

- Complete one instruction every cycle
 - $CPI = 1$

- Speedup is equal to the number of pipeline stages

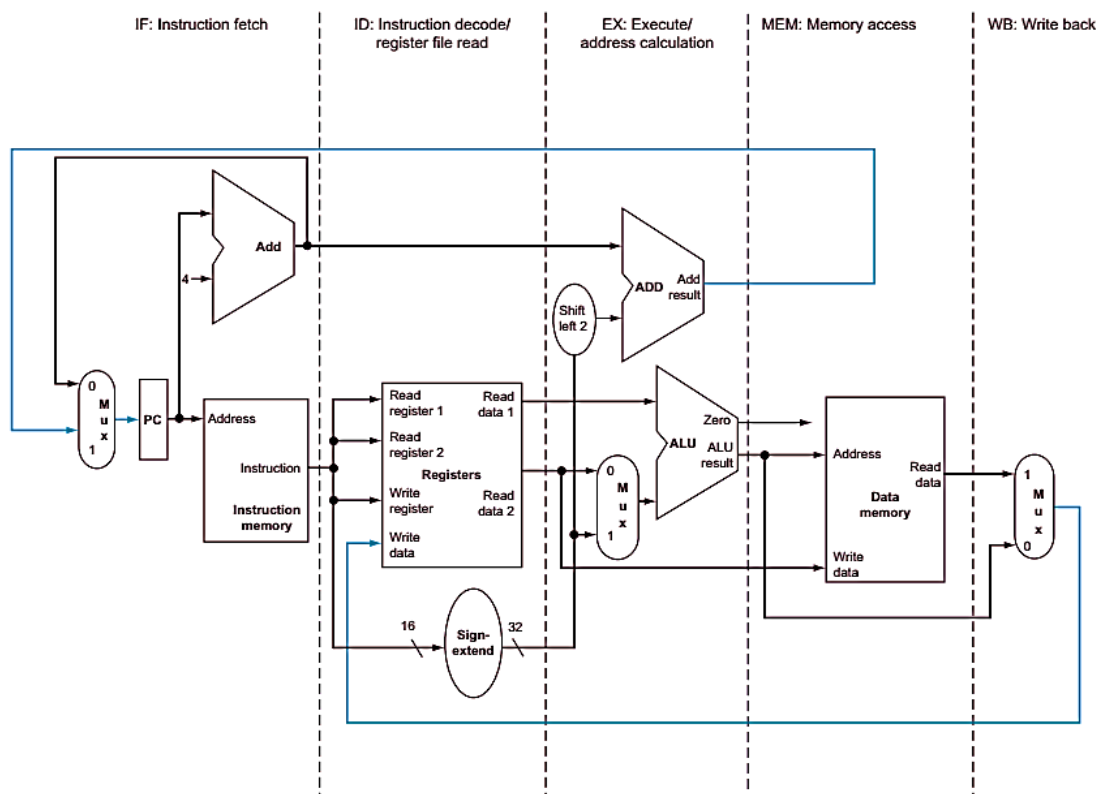


Figure B1.4: Single Cycle Data Path

Figure B1.4 shows the single-cycle datapath with the pipeline stages identified. The division of an instruction into five stages means a five-stage pipeline, which in turn means that up to five instructions will be in execution during any single clock cycle. Thus, we must separate the datapath into five pieces, with each piece named corresponding to a stage of instruction execution:

1. IF: Instruction Fetch
2. ID: Instruction decode and register file read
3. EX: Execution or address calculation
4. MEM: Data memory access
5. WB: Write back

In Figure B1.4, these five components correspond roughly to the way the data-path is drawn; instructions and data move generally from left to right through the five stages as they complete execution. Data flowing from right to left does not affect the current instruction; these reverse data movements influence only later instructions in the pipeline. Note that the first right-to-left flow of data can lead to data hazards and the second leads to control hazards. **(Patterson and Hennessy, pp 312-313 5th Edition)**

B 1.2. Computation of the number of clock cycles with stage delay as 0ns for 5 stage pipeline processor:

Cycle	1	2	3	4	5	6	7	8	9	10	11
INS#1	IF	ID	EXE	MEM	WB						
INS#2		IF	ID	EXE	MEM	WB					
INS#3			IF	ID	EXE	NOP	NOP				
INS#4						IF	ID	EXE	NOP	WB	
INS#5							IF	ID	EXE	NOP	WB

The given set of instructions are performed keeping the assumptions in place, let's see how the instructions are executed,

1. The first instruction is a simple load immediate instruction; hence the flow is regular and there is no stall
2. The second instruction is also a load immediate and does not depend upon the previous instruction.
3. The third instruction is a branch type instruction that is depended on the previous instruction, this is a Read-After-Write Hazard, but since we assumed that the we support Data Forwarding, just after the Execution of the previous instruction the results are forwarded to the inputs of the ALU of the next instruction for execution, this prevents stall of the CPU.
4. The fourth instruction is after a branch instruction, so IF has to wait for the branch execution to complete and then this instruction starts execution.
5. The fifth instruction is also depended upon the previous instruction, this is Read-After-Write Hazard, once again the output from the execution of the previous instruction is forwarded to the ALU of this instruction, this prevents stalling of the CPU.

The total number of clock cycles for execution of the given set of instructions are 11.

Solution to Question No. 2 Part B:**B 2.1. Introduction to split stages:**

Splitting the longest stage is the only way to reduce the cycle time. After splitting it, the new cycle time is based on the new longest stage. What this does is balance the load into two stages of pipeline this will reduce the longest stage into smaller stages and hence reduce the bottleneck. The throughput or the cycle time depends on this longest stage and splitting it distributes the stage evenly.

A 2GHz Pentium® 4 processor has a 500ps cycle time, with a portion of the cycle used for skew, jitter, latching and other pipeline overheads. The cycle time that is not used for pipeline overhead is then dedicated for useful work. Assuming that the pipeline overhead per cycle is 90 ps, one can calculate the total "algorithmic work" associated with the branch misprediction pipeline as the number of stages * useful work/stage, or (20 stages* (500ps - 90ps) = 8200 ps of algorithmic work in branch miss loop).

In these calculations, we have included the communication time in the "useful work" component of the cycle time. The communication time includes the latency of wire delays, and therefore the "useful work" in a path is a function of the floorplan. This is particularly relevant in areas such as the branch misprediction loop, where the latency of driving the misprediction signal from the branch resolution unit to the front end becomes a key component of the overall loop latency. If we assume that the overhead per cycle is constant in a given circuit technology, in this case 90 ps, we can increase the processor frequency by reducing the "useful time" per cycle. As the useful time per cycle approaches zero, the total cycle time approaches the "overhead time". (Eric Sprangle, Intel)

ID Stage is split into ID1 and ID2 stages; EXE is split into EXE1 and EXE2

B 2.2 Compute the number of clock cycles with stage delay as 0ns including pipeline bubbles for 7 stage pipeline processor

Cycle	1	2	3	4	5	6	7	8	9	10	11
INS#1	IF	ID1	ID2	EXE1	EXE2	MEM	WB				
INS#2		IF	ID1	ID2	EXE1	EXE2	MEM	WB			
INS#3			IF	STL	STL	ID1	ID2	EXE1	EXE2	NOP	NOP
INS#4										IF	ID1
INS#5											IF

Cycle	12	13	14	15	16	17	18	19
INS#1								
INS#2								
INS#3								
INS#4	ID2	EXE1	EXE2	NOP	WB			
INS#5	STL	STL	ID1	ID2	EXE1	EXE2	NOP	WB

Explanation:

1. The first instruction is independent of any other and is executed normally without any stalls, the ID and EXE stage are split into two stages.
2. The second instruction is independent of the previous instruction and executes normally.
3. The third instruction is a branch instruction that depends on the previous instruction's output for its execution, hence after the IF stage the CPU is stalled, the Output obtained in the EXE2 stage of the previous instruction is forwarded into this instruction's ID1 stage. Since the branch instruction does not have a MEM and WB, we insert a NOP bubble into it for a uniform 7-Stage pipeline.
4. The fourth instruction is after the Branch instruction and is dependent on its execution, hence the IF stage of this starts after the execution of the previous instruction. This instruction does not have a MEM stage and hence we introduce a NOP bubble to complete the uniform 7-Stage pipeline.
5. The fifth instruction is dependent on the previous instruction and the CPU is stalled until the Execution of the previous instruction, the output is then forwarded to the ID1 stage of this instruction and after that the flow continues as usual.

Hence the total number of Clock Cycles for the execution of this would be 19.

B 2.3 Analyze the performance of 5 stage and 7 stage pipelined processor

The two different types of processors can be compared in terms of their throughput,

1. 5-Stage Processor

Cycle Time = Stage with the longest delay = 400ps

Instruction Count = 5

Time taken for the execution of 5 instructions:

$$\text{Time Taken} = (n - 1) + \text{Instructions Count} \times \text{Cycle Time}$$

$$\text{Time Taken} = ((5 - 1) + 5) \times 400\text{ps} = 3600\text{ps}$$

Throughput is defined as

$$\text{Throughput} = \frac{\text{Instruction Count}}{\text{Total time taken}}$$

Hence,

$$\text{Throughput} = \frac{5}{3600 \times 10^{-12}} = 1.38888 \times 10^9 \text{ Instructions/sec}$$

2. 7-Stage Processor

Assuming that the stage delay is equally split when the stage is split, i.e. EXE which takes 400ps when split into EXE1, EXE2 takes 200ps, 200ps respectively, same is applicable for ID stage, which is split into ID1, ID2 and the stage delay is then split into 100ps, 100ps respectively.

Cycle Time = Stage with the longest delay = 200ps

Instruction Count = 7

Time taken for the execution of 7 instructions:

$$\text{Time Taken} = (n - 1) \times \text{Cycle Time} + \text{Instructions Count} \times \text{Cycle Time}$$

$$\text{Time Taken} = ((7 - 1) + 5) \times 200\text{ps} = 2200\text{ps}$$

Throughput is defined as

$$\text{Throughput} = \frac{\text{Instruction Count}}{\text{Total time taken}}$$

Hence,

$$\text{Throughput} = \frac{5}{2200 \times 10^{-12}} = 2.272727 \times 10^9 \text{ Instructions/sec}$$

Throughput Comparison:

Comparing the two types of processor based on their throughput it can be observed that the 7-Stage processor can execute more instructions than the 5-stage processor in the same amount of time, i.e. the instructions per second for the 7-Stage processor is more.

The two processors can also be compared based on their CPI

1. 5-Stage Processor

Number of Clock Cycles = 11

Instructions Count = 5

CPI is given by,

$$CPI = \frac{\text{Number of Clock Cycles}}{\text{Total Number of Instructions}}$$

$$CPI = \frac{11}{5} = 2.2 \text{ cc/ins}$$

where cc = clock cycles, ins = instructions

2. 7-Stage Processor

Number of Clock Cycles = 19

Instructions Count = 5

CPI is given by,

$$CPI = \frac{\text{Number of Clock Cycles}}{\text{Total Number of Instructions}}$$

$$CPI = \frac{19}{5} = 3.8 \text{ cc/ins}$$

where cc = clock cycles, ins = instructions

CPI (Clock Cycles per Instruction) Comparison:

Comparing the two processors based on their CPI, it can be seen that the 5-Stage Processor has lower CPI compared to the 7-Stage Processor.

B 2.4 Based on the analysis, recommend the best design for high performance computing applications:

From the comparisons made in B2.3, it's clear that the 7-Stage processor has more performance capabilities than the 5-Stage Processor, it can execute more instructions in less time.

High-performance computing (HPC) is the use of super computers and parallel processing techniques for solving complex computational problems. HPC technology focuses on developing parallel processing algorithms and systems by incorporating both administration and parallel computational techniques.

A HPC for small business could have a few as four nodes, or 16 cores, a common cluster size in many business is between 16 and 64 nodes, or from 64 to 256 cores, these types of parallel computing requires a good processor design to perform parallel operations with greater efficiency, to choose from the 5-Stage and the 7-Stage processor, the 7-Stage processor would be preferred here, it would increase the number of instructions being executed per second and hence therefore increasing the efficiency of whole system.

B 2.5 Conclusion:

From the observations made from the two kinds of processors, the second one in which the stages are split into 2, the performance of the processor increases, since the throughput of the processor increases, one tradeoff of such a design is that the CPI of the processor is increased, although the number of instructions executed per second is increased.

It can also be concluded that comparing just the CPI of the two processors does not give a clear idea of how the processor will function under full load, the throughput of the processor matters since it tells the

number of instructions that the processor can execute when in load. There are other factors also that need to be considered along with these parameters.

1. Patterson and Hennessy, Computer Organization 5th Edition.
2. Increasing Processor Performance by Implementing Deeper Pipelines, Eric Sprangle, Doug Carmean, Intel Corporation.
3. <http://parallelcomp.uw.hu/ch01lev1sec2.html>