COMPUTER ARCHITECTURE
ASSIGNMENT NO. 1, Fall 2019
Solution

Q1: What is the clock frequency of a processor that executes a program of 60,000 instructions in 4 microseconds? The average clock cycles per instruction of the processor is 1.5.

IC = 60,000

Execution time = 4 x10^{-6} sec

Clock frequency =?

$$\text{CPU Time} = \frac{IC \times CPI}{CLK\ Freq}$$

$$CLK\ Freq = \frac{IC \times CPI}{CPU\ Time}$$

$$CLK\ Freq = \frac{60{,}000 \times 1.5}{4 \times 10^{-6}}$$

Clock Frequency = 22.5 GHZ

Q2: Calculate the CPI for a processor that executes a 30,000 instructions program in 18 microseconds, if the clock frequency is 6 GHz.

$$CPI = ?$$

$$IC = 30{,}000$$

$$CPUTime = 18 \times 10^{-6} Sec$$

$$Clock\ Frequency = 6 \times 10^{9} Hz$$

$$CPI = \frac{CPUtime\ \times clock\ rate}{IC}$$

$$CPI = \frac{18 \times 10^{-6}\ \times 6 \times 10^{9}}{30{,}000}$$

$$CPI = 3.6$$

Q3: Calculate the CPI of a processor using the data available in the following table:

Type of Instruction	Frequency of Occurrence	No. of clock cycles
ALU	30%	2
Load/Store	25%	4
Control flow	18%	3
Others	27%	3

$$CPI_{total} = (0.3 \times 2) + (0.25 \times 4) + (0.18 \times 3) + (0.27 \times 3)$$
$$CPI_{total} = 0.6 + 1 + 0.54 + 0.81$$
$$CPI = 2.95$$

Q4: A processor is enhanced by adding a graphics unit. A speedup of 15 is achieved for this enhanced mode. If the overall speedup is observed to be 2 for an application that runs on the enhanced processor, what is the percentage of graphics instructions in the example application?

$$Speedup_{enhanced} = 15$$

$$Speedup_{overall} = 2$$

$$F_{enhanced} = ?$$

$$Speedup_{enhanced} = \frac{1}{(1 - F_{enhanced}) + \frac{F_{enhanced}}{Speedup_{enhanced}}}$$

$$2 = \frac{1}{(1 - F_{enhanced}) + \frac{F_{enhanced}}{15}}$$

$$\frac{15(1 - F_{enhanced}) + F_{enhanced}}{15} = \frac{1}{2}$$

$$\frac{(15 - 14F_{enhanced})}{15} = \frac{1}{2}$$

$$2 \times (15 - 14F_{enhanced}) = 15$$

$$(30 - 15) = 28F_{enhanced}$$

$$F_{enhanced} = \frac{15}{28} = 0.5357$$

53.57%

Q5: Designers are given two alternatives to choose from while enhancing the performance of a processor: Enhance the entire ALU to get an average speedup of 5 in the ALU instructions or enhance ADD/SUB instructions to get a speedup of 10. ALU instructions constitute 40% of the total instruction and ADD/SUB instructions constitute 25% of the total. Evaluate both alternatives to select the better option.

1. For ALU

$$Speedup_{enhanced} = \frac{1}{(1 - F_{enhanced}) + \frac{F_{enhanced}}{Speedup_{enhanced}}}$$

$$Speedup_{enhanced} = \frac{1}{(1 - 0.4) + \frac{0.4}{5}}$$

$$Speedup_{enhanced} = \frac{1}{0.6 + 0.08} = \frac{1}{0.68} = 1.47$$

2. For ADD/SUB

$$Speedup_{enhanced} = \frac{1}{(1 - F_{enhanced}) + \frac{F_{enhanced}}{Speedup_{enhanced}}}$$

$$Speedup_{enhanced} = \frac{1}{(1 - 0.25) + \frac{0.25}{10}}$$

$$Speedup_{enhanced} = \frac{1}{(0.75) + 0.025} = \frac{1}{0.775}$$

$$Speedup_{enhanced} = 1.2903$$

Enhancement the entire ALU gives better results compared to just enhancing ADD/SUB instructions.

Q6: A non-pipelined processor has 4 steps of execution, where the time taken is 30nsec, 25nsec, 35nsec and 40nsec for the respective steps. The processor is pipelined by adding pipeline latches between steps of execution. If the latch delay is 2nsec, what is the pipeline clock frequency? Calculate the speedup of the pipelined processor over its non-pipelined counterpart? Is it possible to achieve this speedup? Why or why not?

Solution-

Given-

Four stage pipeline is used

Delay of stages = 30, 25, 35 and 40 ns

Latch delay or delay due to each register = 2 ns

Pipeline Cycle Time-

Cycle time

= Maximum delay due to any stage + Delay due to its register

= Max { 30, 25, 35, 40 } + 2 ns

= 40 ns + 2 ns

= 42 ns

$$\text{Clock frequency } = \frac{1}{Clock\ cycle}$$

$$\text{Clock frequency } = \frac{1}{42 \times 10^{-9}} = 2.38 \times 10^{7}$$

$$\boldsymbol{Clock\ frequency} = \mathbf{23.87}\ \boldsymbol{MHZ}$$

Non-Pipeline Execution Time-

Non-pipeline execution time for one instruction

= 30 ns + 25 ns + 35 ns + 40 ns

= 130 ns

Speed Up Ratio-

Speed up = Non-pipeline execution time / Pipeline execution time

= 130 ns / 42 ns

Speed up = 3.095

Q7: Identify all data hazards in the following code segment. Give solutions through internal forwarding in a RISC V processor. Show the space time diagram of the code after all hazards are removed/optimized.

1.	**LD**	**x1, 8(x2)**
2.	**ADD**	**x3, x1, x5**
3.	**SUB**	**x6, x1, x5**
4.	**ADD**	**x4, x3, x6**
5.	**OR**	**x5, x4, x7**
6.	**SD**	**x5, 8(x2)**

Hazards:

1. Instruction 2 : Reg x_1 RAW hazard as ADD instruction reads x1 before load writes it.
2. Instruction 3: Reg x_1 RAW hazard, as SUB instruction reads x1 before load writes it.
3. Instruction 4: Reg x_3, x_6 RAW hazards, as ADD instruction read x3 and x6 before instruction 2, and 3 writes them respectively.
4. Instruction 5: x4 RAW hazard, as OR instruction reads it before ADD instruction writes it.
5. Instruction 6: x_5 WAW hazard, as instruction SD writes on x_5 before instruction OR writes on x_5.

Solution:

1. The **LD** instruction **does not** have the data until the end of clock cycle 4 (**MEM**) , while the **ADD** (2nd instruction) instruction needs to have the data by the beginning of that clock cycle (**EX$_{add}$**). In this case one stall is needed and after that the value of load can directly be forwarded to ALU (EX$_{add}$) from MEM/WEB register.
2. For **SUB** instruction (3rd instruction) we can forward the result immediately to the ALU (**EX$_{sub}$**) from the MEM/WB register(**MEM**).
3. For next **ADD** instruction (4rth Instruction) we can forward the result of 3rd instruction directly to ALU(**EX$_{add}$**) from the **EX/MEM** register.
4. For **OR** instruction (5th) again use forwarding result of ADD instruction (4rth) in directly forwarded to ALU(**EX$_{or}$**) from EX/MEM register .
5. In **SD** (6th Instruction) we can directly forward result of OR from **EX/MEM** register to the **MEM** stage.

		C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	
1	LD x1, 8(x2)	IF	ID	EX	ME	WB						
	Stall											
2	ADD x3, x1, x5			IF	ID	EX	ME	WB				
3	SUB x6, x1, x5				IF	ID	EX	ME	WB			
4	ADD x4, x3, x6					IF	ID	EX	ME	WB		
5	OR x5, x4, x7						IF	ID	EX	ME	WB	
6	SD x5, 8(x2)							IF	ID	EX	ME	WB

Q8: What is a delayed branch technique of handling control hazards? Give an example to illustrate using RISC V and the five stage integer pipeline.

One way to maximize the use of the pipeline, is to find an instruction that can be safely executed whether the branch is taken or not, and execute that instruction. So, when a branch instruction is encountered, the hardware puts the instruction following the branch into the pipe and begins executing it, just as in predict-not-taken. However, unlike in predict-not-taken, we do not need to worry about whether the branch is taken or not, we do not need to clear the pipe because no matter whether the branch is taken or not, we know the instruction is safe to execute. How do we know? The compiler promised it would be safe. This technique is called delayed branch technique. Following table shows how delayed branch works either branch is taken or not taken.

Untaken branch instruction	IF	ID	EX	MEM	WB				
Branch delay instruction $(i+1)$		IF	ID	EX	MEM	WB			
Instruction $i+2$			IF	ID	EX	MEM	WB		
Instruction $i+3$				IF	ID	EX	MEM	WB	
Instruction $i+4$					IF	ID	EX	MEM	WB
Taken branch instruction	IF	ID	EX	MEM	WB				
Branch delay instruction $(i+1)$		IF	ID	EX	MEM	WB			
Branch target			IF	ID	EX	MEM	WB		
Branch target + 1				IF	ID	EX	MEM	WB	
Branch target + 2					IF	ID	EX	MEM	WB