Ira A. Fulton Schools of Engineering
School of Computing, Informatics and Decision Systems Engineering

SER 450 **COMPUTER ARCHITECTURE**

Your Name: Jacob Hreshchyshyn Jacob Hreshchyshyn

When completing this worksheet, please use a different color text or typeface so that we can easily identify your work. Remember to show your work / give justification for your answers.

Portions of this material are derived from "Computer Organization and Design", Patterson & Hennessy.

PRACTICE PROBLEMS 1

Problem 1:

The eight great ideas in computer architecture are similar to ideas from other fields. Match the eight ideas from computer architecture , "Design for Moore's Law", "Use Abstraction to Simplify Design", "Make the Common Case Fast", "Performance via Parallelism", "Performance via Pipelining", "Performance via Prediction", "Hierarchy of Memories", and "Dependability via Redundancy" to the following ideas from other fields:

A. Assembly lines in automobile manufacturing

B. Suspension bridge cables

C. Aircraft and marine navigation systems that incorporate wind information

D. Express elevators in buildings

E. Library reserve desk

F. Increasing the gate area on a CMOS transistor to decrease the switching time

G. Adding electromagnetic aircraft catapults (which are electrically-powered as opposed to current steam-powered models), allowed by the increased power generation offered by the new reactor technology

H. Building self-driving cars whose control systems partially rely on existing sensor systems already installed into the base vehicle, such as lane departure systems and smart cruise control systems

A can be matched with Performance via Pipelining because of the practice of chaining together multiple processes to accomplish a task efficiently. The textbook uses the example of a fire brigade creating a chain of people that passes buckets of water up and down the chain to quickly assist in extinguishing a fire and avoid wasting time by running back and forth to get water and dump it on the fire.

B can be matched with Dependability via Redundancy because there exist multiple cables designed to support the bridge. Because there are so many cables that support the bridge, the

bridge is more likely to remain intact should a limited number of cables fail or if certain cables become deteriorated over time. This reflects the Dependability via Redundancy great idea, which is concerned with the creation of systems that include "redundant components that can take over when a failure occurs and to help detect failures" (Required Reading 12).

C can be matched with Performance via Prediction because of the process of making an educated guess about where to direct the airplane using wind information in order to potentially avoid wasting more time when encountering unforeseen headwinds.

D can be matched with Make the Common Case Fast because express elevators are designed to quickly deliver passengers to specific floors in a building. A good use of this design would be to deliver passengers only to floors that the majority of the passengers visit. In other words, the elevator ought to be able to quickly deliver passengers to the most commonly visited floors in order to aid in overall efficiency.

E can be matched with Hierarchy of Memories because of the reserve desk's function of making information quickly available to those who are most likely to need it. However, the rest of the library helps to illustrate the Hierarchy of Memories since it would be more time consuming to retrieve the particular piece of information one is searching for by exploring the rest of the library, especially if that information is more dense.

F can be matched with Design for Moore's Law because it closely reflects the idea of circuit resources doubling every 18-24 months to improve performance.

G can be matched with Performance via Parallelism because of the use of the improved reactor technology in launching the aircraft electrically working in parallel with the thrusters on the aircraft themselves to effectively launch the aircraft.

H can be matched with Use Abstraction to Simplify Design because of the self-driving vehicle's systems that remove more details associated with operating the vehicle to simplify the means to use it.

Problem 2:

Describe the steps that transform a program written in a high-level language such as C into a representation that is directly executed by a computer processor.

The first step is to allow the compiler to take the code written in a high-level language and translate the code into assembly code. The next step is to allow the assembler to take the symbolic assembly code and translate it into binary machine language, which is a collection of 1's and 0's. Such a collection is a fitting representation of on and off, which a computer processor can easily use to execute instructions electrically.

Problem 3:

Consider three different processors P1, P2, and P3 executing the same instruction set. P1 has a 3 GHz clock rate and a CPI of 1.5. P2 has a 2.5GHz clock rate and a CPI of 1.0. P3 has a 4.0 GHz clock rate and a CPI of 2.2.

A. Which processor has the highest performance expressed in instructions per second?

One can find the instructions per second of each processor by dividing the clock rate (cycles per second) of each processor by their corresponding CPI's (cycles per instruction).

P1 instructions per second = 3.0 GHz/1.5 = $2.0 * 10^9$

P2 instructions per second = 2.5 GHz/1.0 = $2.5 * 10^9$

P3 instructions per second = 4.0 GHz/2.2 = $1.8 * 10^9$

Of these, P2 has the highest number of instructions per second, meaning that P2 has the highest performance.

B. If the processors each execute a program in 10 seconds, find the number of cycles and the number of instructions for each processor.

To find the number of cycles for each processor, we can multiply the clock rate (cycles per second) by the execution time (seconds).

P1 cycles = 3.0 GHz * 10 s = $3.0 * 10^{10}$

P2 cycles = 2.5 GHz * 10 s = $2.5 * 10^{10}$

P3 cycles = 4.0 GHz * 10 s = $4.0 * 10^{10}$

To find the number of instructions for each processor, we can multiply the previously found instructions per second of each processor by the execution time.

P1 instructions = $2.0 * 10^9 * 10 = 2.0 * 10^{10}$

P2 instructions = $2.5 * 10^9 * 10 = 2.5 * 10^{10}$

P3 instructions = $1.8 * 10^9 * 10 = 1.8 * 10^{10}$

C. We are trying to reduce the execution time by 30% but this leads to an increase of 20% in the CPI. What clock rate should we have to get time reduction?

We know that the current Execution Time (or CPU Time) = (Instruction Count * CPI)/Clock Rate. We also know that a 30% reduced execution time, which is 10 s – 10 s * (30 / 100), is 7 s. We also know that a 20% increased CPI, which is CPI + CPI * (20 / 100).

If we rewrote the Execution Time equation to find the Clock Rate, we would have Clock Rate = (Instruction Count * New CPI)/7 s. Now we plug and chug.

P1 New Clock Rate = ($2.0 * 10^{10}$ instructions * (1.5 + 1.5 * (20/100))/7 s = $5.14 * 10^9$ Hz or 5.14 GHz.

P2 New Clock Rate = ($2.5 * 10^{10}$ instructions * (1.0 + 1.0 * (20/100)))/7 s = $4.29 * 10^9$ Hz or 4.29 GHz.

P3 New Clock Rate = ($1.8 * 10^{10}$ instructions * (2.2 + 2.2 * (20/100))) 7 s = $6.79 * 10^9$ Hz or 6.79 GHz.

Problem 4:

Compilers can have a profound impact on the performance of an application. Assume that for a program, compiler A results in a dynamic instruction count of 1.0E9 and has an execution time of 1.1 seconds, while compiler B results in a dynamic instruction count of 1.2E9 and an execution time of 1.5 seconds.

A. Find the average CPI for each program given that the processor has a clock cycle time of 2.0 ns (2e-9 seconds).

We know that the CPU Time = Instruction Count * CPI * Clock Cycle Time. So, to find the CPI, we can rewrite the equation as CPI = CPU Time/(Instruction Count * Clock Cycle Time).

Compiler A CPI = 1.1 s / ($1.0 * 10^9 * 2.0 * 10^{-9}$ s) = 0.55

Compiler B CPI = 1.5 s / ($1.2 * 10^9 * 2.0 * 10^{-9}$ s) = 0.625

B. Assume the compiled programs run on two different processors. If the execution times on the two processors are the same, how much faster is the clock of the processor running compiler A's code versus the clock of the processor running compiler B's code?

To compare the two processors, we can compare their clock rates. We know that CPU Time = Instruction Count * CPI * Clock Cycle Time. Clock Cycle Time is another way of describing the duration of a clock cycle, or a clock period. The inverse of this is the Clock Frequency or Clock Rate. So, we can start by rewriting the above equation to solve for Clock Cycle Time, which would look like Clock Cycle Time = CPU Time/(Instruction Count * CPI). The inverse of both sides of this equation would give us Clock Frequency = (Instruction Count * CPI)/CPU Time.

Compiler A Clock Frequency = CPU Time/($1.0 * 10^9 * 0.55$)

Compiler B Clock Frequency = CPU Time/($1.2 * 10^9 * 0.625$)

Since we know that Compiler A will be faster than Compiler B, we can determine how much faster by dividing the Clock Frequency of B by A. Since the CPU Times are the same between the two processors, those will cancel out, allowing us to perform the following division:

($1.2 * 10^9 * 0.625$)/($1.0 * 10^9 * 0.55$) = 1.37. Thus, A is 1.37 times faster than B.

C. A new compiler is developed that uses only 6.0E8 instructions and has an average CPI of 1.1. What is the speedup of using this new compiler versus using compiler A or B on the original processor from part A?

We know that CPU Time = Instruction Count * CPI * Clock Cycle Time. Since Clock Cycle Time is determined by the hardware and the comparison between compilers is presumably happening on the same processor, the Clock Cycle Time will cancel out. Additionally, we know that the new compiler gives us a performance increase, which means that the faster CPU Time will appear in the denominator in our division of CPU times.

CPU Time A/New CPU Time = ($1.0 * 10^9 * 0.55$ * Clock Cycle Time)/($6.0 * 10^8 * 1.1$ * Clock Cycle Time) = 0.83. Thus, the new compiler is 0.83 times faster than Compiler A.

CPU Time B/New CPU Time = ($1.2 * 10^9 * 0.625$ * Clock Cycle Time)/($6.0 * 10^8 * 1.1$ * Clock Cycle Time) = 1.14. Thus, the new compiler is 1.14 times faster than Compiler B.

Problem 5:

The text cites as a pitfall the utilization of a subset of the performance equation as a performance metric. To illustrate this, consider the following two processors: P1 has a clock rate of 4 GHz, average CPI of 0.9, and requires the execution of 5.0E9 instructions. P2 has a clock rate of 3.0 GHz, an average CPI of 0.75, and requires 1.0E9 instructions

The usual fallacy is to consider the computer with the largest clock frequency as the fastest – use the computer performance model to check if this is true.

P1 Time = $(5.0*10^9$ instructions * 0.9)/$(4.0 * 10^9$ Hz) = 1.125 s

P2 Time = $(1.0*10^9$ instructions * 0.75)/$(3.0 * 10^9$ Hz) = 0.25 s

Thus, despite P2 having a lower clock rate (3 GHz as opposed to 4 GHz), P2 has the better performance. This is in large part due to the lower number of instructions executed on P2 and the lower CPI of P2.

Problem 6:

Assume a program requires the execution of 50E6 floating-point (FP) instructions and 110E6 INT (integer) instructions, 80E6 L/S (load/store) instructions, and 16E6 branch instructions. The CPI for each type of instruction is 1, 1, 4, and 2 respectively. Assume that the processor has a 2.0 GHz clock rate.

By how much must we improve the SPI of the FP instructions if we want the program to run two times faster?

Before answering the question, I should address what I believe is a typo. Instead of SPI, I believe the problem meant to ask about the CPI. The CPI improvement is what this answer will address.

We know that, from the example provided in the video lecture, we can find the number of clock cycles in this sequence of program executions by multiplying each program's CPI with their Instructions Counts and add them up.

Clock Cycles = $(1 * 50 * 10^6) + (1 * 110 * 10^6) + (4 * 80 * 10^6) + (2 * 16 * 10^6) = 5.12 * 10^8$

We can use this to find the CPU Time, which can be found by dividing the Clock Cycles by the Clock Rate, which is provided in the problem.

CPU Time = $(5.12 * 10^8)/(2.0 * 10^9) = 0.256$ s.

This is the time to beat two times over. In other words, we need to attempt to improve the FP CPI to achieve a CPU Time of 0.256 s / 2 = 0.128 s.

We can combine the Clock Cycles equation with the CPU Time equation to solve for the FP CPI.

0.128 s = $((\text{FP CPI} * 50 * 10^6) + (1 * 110 * 10^6) + (4 * 80 * 10^6) + (2 * 16 * 10^6))/(2.0 * 10^9)$

If we separated the ratio of the product of the FP CPI and FP Instruction Count to Clock Rate from the rest of the sum and 0.128 s from that same sum, we would get the following:

$(\text{FP CPI} * 50 * 10^6)/(2.0 * 10^9) = -0.103$ s.

Thus, we can see that, no matter how we manipulate this expression, there is no way to find a valid, non-negative FP CPI, meaning that it is impossible to engineer the program to run twice as fast by improving the FP CPI.