আন্তর্জাতিক ইসলামী বিশ্ববিদ্যালয় চট্টগ্রাম
الجامعة الإسلامية العالمية شيتاغونغ
International Islamic University Chittagong

ASSIGNMENT

Department : Computer Science & Engineering
Assignment On : The role of performance, Language of Machine
Course Name : Computer Architecture
Course Code : CSE- 3521

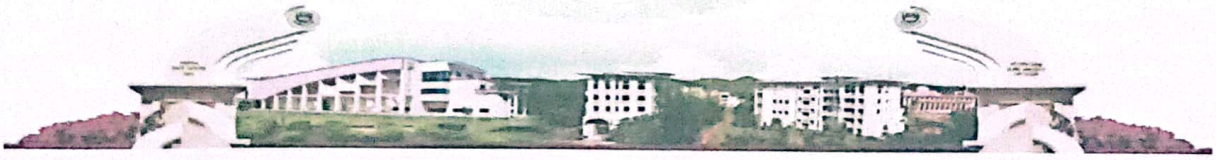
Submitted To Mrs. Sanjida Sharmin
Lecturer

Submitted By

Name : Farzana Yeasmin Etu
ID No : C223249
Semester : 5th
Section : 5BF

Date of Issue : 19.09.2024
Date of Submission : 28.09.2024

Remark



Answer to the question no. 01

ⓐ Response time: The time between the start and completion of a task is called response time or execution time.

Throughput: The total amount of work done in a given time is called throughput

CPI (clocks per Instruction): CPI is a key metric used to measure the performance of a processor. It represents the average number of clock cycles a processor takes to execute a single instruction.

It can be expressed using the formula

$$CPI = \frac{\text{CPU clock rates}}{\text{Instruction count}}$$

(b) Which code sequence executes most instruction.

Here, $X = 9$

For, Compiler I = A+B+C = 4+2+3 = 9

For compiler 2 = A+B+C = 7+6+9 = 22

∴ Compiler 2 execute most instruction that is 22.

* Which will be faster?

We know, CPU clock cycle = $\Sigma (CPI_i \times C_i)$

$CPU_1 = (4 \times 1) + (2 \times 2) + (3 \times 3) = 17$

$CPU_2 = (7 \times 1) + (6 \times 2) + (9 \times 3) = 46$

∴ So code sequence 1 ~~with~~ is faster.

* What is the CPI for each instruction?

$$CPI_1 = \frac{CPU_1 \text{ clock Rate}}{\text{Instruction Count}_1} = \frac{17}{9} = 1.89$$

$$CPI_2 = \frac{CPU_2 \text{ clock Rate}}{\text{Instruction Count}_2} = \frac{46}{22} = 2.09$$

(c) Given, Unix time command 80.4u 10.5s 2:40

User CPU time 80.4 sec.

System CPU time 10.5 sec

Total time $= (2 \times 60 + 40) = 160$ sec.

Percentage according to total elapsed time $= \frac{80.4 + 10.5}{160} = 0.56 = 56\%$.

Answer to the question no- 2

@ Instructions: The words of a machine's language are called instructions. Example: MOV, ADD, SUB.

Major differences between the variables of a programming language and registers:

Points	Variables	Registers
1. Level of Abstraction	1. Variables are high-level abstraction in programming languages	1. Registers are low-level hardware components in the CPU.
2. Storage location	2. Variables are store in memory (RAM)	2. Registers are stored in CPU itself.

3. Purpose and Use	3. Variable store data or reference used by a program.	3. Register hold data that the CPU is working with
4. Size	4. Variables can represent much larger data structures.	4. Registers are very small like 32-bit, 64-bit
5. Speed	5. Variables are slower to access due to the need to fetch them from memory	5. Registers are extremely faster ~~fast~~ because they are part of processor.

(b) The design principle "smaller is faster" in MIPS refers to the idea that smaller components, such as fewer registers, simple instruction sets, and smaller hardware circuits, can operate more quickly than larger, more complex components. This principle focuses on keeping the CPU

design lean, ensuring faster execution and higher efficiency.

Example: In MIPS, basic instructions like ADD or LOAD operate with just a few operands. Each of these instructions can be executed in a single cycle because they are simple. ~~Confl~~ Contrast this with CISC architectures like x86, where some instructions are most complex and take multiple cycles to execute.

(c) (i) add $s1, $t5, $t6

This is an R-type instruction because it's an arithmetic operation between register

op code (6 bits)	rs (5 bits)	rt (5 bits)	rd (5 bits)	shamt (5 bits)	funct (6 bits)
~~01011~~ 000000	~~01100~~ 01011	01100	10001	000000	100000

Binary encoding to final 32-bit machine code

0000 0001 0110 1100 1000 1000 0010 0000

(ii) bne $s2, $s4, 100

This is an I-type instruction because it uses an immediate value and a branch operation

op code	rs	rt	immediate (16 bits)
000101	10010	10100	0000 0000 0110 0100

Final 32-bit machine code

0001 0110 0101 0100 0000 0000 0110 0100

(iii) sw $s1, 100($s2)

This is also an I-type instruction because it uses an immediate offset and a memory operation.

op code	rs	rt	immediate (16 bits)
10111	10010	10001	0000 0000 0110 0100

Final 32-bit machine code

1010 1110 0101 0001 0000 0000 0110 0100