Started on	Saturday, 26 September 2020, 2:03 PM
State	Finished
Completed on	Saturday, 26 September 2020, 3:22 PM
Time taken	1 hour 18 mins
Grade	Not yet graded

Question **1**
Complete
Marked out of 5.00

Consider a four-stage pipeline for instruction execution in scalar processor. The four stages of the pipeline are *Fetch:* to fetch the instruction from memory, *decode:* to fetch operands from register file, *Execute/Memory* which performs the execution of instruction or memory operation such as Load/Store and finally the *Writeback* stage which writes the result back to a register file.

a) What is data dependency in pipeline? Explain how it it can resolved? **[5 Marks]**

data dependency.jpg

Question **2**
Complete
Marked out of 5.00

What is control dependency in instruction pipeline? Explain how to resolve it using branch prediction? **[5 Marks]**

control dependency.jpg

Question **3**

Not answered

Marked out of 10.00

Consider the following program to be executed in the four-stage pipeline. Assume each stage takes one clock cycle for execution. Compute total number of clock cycles required to execute this program in pipeline by considering the dependencies of the instruction. Compare the ratio of improvement with respect to sequential execution. *[note: No need to draw complete stages of pipeline for each instruction execution. But mention what instruction gets executed in each stage for every clock cycle.]* [**10 Marks]**

Program to find largest of three numbers

MOV SI, 4500H

MVI BL, 06H

MVI CL, 15H

MVI DL, 25H

CMP BL, CL

JNC NEXT1

MOV BL, CL

NEXT1: CMP BL, DL

JNC NEXT2

MOV BL, DL

NEXT2: MOV [SI], BL

HLT

Question **4**

Complete

Marked out of 10.00

Consider a shared memory model, where 3 processors contain a separate cache and all the processors are connected through a common bus to main memory. Consider that a variable X which is shared among all the processors and currently value of X is 15 in all the cache and main memory. If the write back policy is used to update the main memory, mention the state of variable X in each processor's cache P1, P2 and P3 for following operations **carried out in sequence**. Assume that MSI protocol is used for memory consistency.
[10 Marks]

(i) P1 reads X

(ii) P2 writes to X and changes it to X=20.

(iii) P3 reads X

Initially all three processor cache lines are in shared state.
all have x=15. main memory has x = 15

a) When P1 reads X it remains in **SHARED** state.

All the cores remain in shared state.

P1 = P2 = P3 = SHARED

b) When P2 writes to X and changes it to X=20.

P2s cache line's tag is changed from **SHARED** to modified.

write back policy causes the main memory to be updated with X's new value.

and because all the cores are snooping the bus, tag lines of **P3** and **P1** are changed to **INVALID.**

P2's cache line is changed to **MODIFIED.**

P2 = MODIFIED **P3 = P1 = INVALID**

c) When **P3** tries to read X , it is not allowed to read from its cache line as it is set to **INVALID**. it has to be read from memory.

Now **P3** and **P2** are set to **SHARED.** whereas **P1** remains **INVALID**

P3 = P2 = SHARED **P1 = INVALID**

MSI.jpg

Question **5**

Complete

Marked out of 10.00

Assume that a city is deployed with 1000 sensor nodes to sense the environment and a system collects the details from each sensor for every hour and stores it in two-dimensional array. The rows indicate sensors and columns indicates the temperature sensor value in 2D array. Write an OpenMP program to calculate the average temperature in each sensor and overall average temperature in that city in a particular day. **[10 marks]**

I have maintained an array with **size = NUMBER OF SENSORS** to record the average temperature for each of the sensors.
I have maintained a **overall_temperature** variable that will require the average overall temperature.

I have used reduction to calculate average overall temperature as there is loop carry dependency that cannot be eliminated

```
#include<stdio.h>
#include<omp.h>
#include<stdlib.h>
#define NUMBER_SENSORS 1000
int main() {
double temperature[NUMBER_SENSORS][24];

srand(0);

for(int i=0; i<1000; ++i) {
for(int j=0; j<24; ++j) {
temperature[i][j]= rand() % 50;
}
}
double overall_average = 0.0;
double average[NUMBER_SENSORS] = {0.0};

#pragma omp parallel for reduction(+:overall_average)
for(int i=1; i<NUMBER_SENSORS; ++i)
{
for(int j=0; j<24; ++j) {
int sum = temperature[i][j];
average[i] += sum;
}
average[i] = average[i] / 24;
overall_average += average[i];
}

for(int i=0; i<100; ++i) {
printf("%f\n", average[i]);
}
overall_average = overall_average / NUMBER_SENSORS;

printf("%f", overall_average);
}
```

program.c.png

◄ Amdahl's Law

Jump to...