CSE-4001 Parallel & Distributed Computing

Slot : A2.

Digital Assignment - I

Name : Ankit Bose Tagore Reg. No : 20BCE2150.

Q1. Develop an execution schedule for given code fragment for adding list of four numbers using a way super scalar processor.

```
Load R1, @2000
add R1, @2004
add R1, @2008
add R1, @200C
Store R1, @3000.
```

Sol^n: To develop an execution schedule for the given code fragment using a-way ~~spectacular~~ superscalar processor and a 5 stage instructional pipeline, we need to consider the pipeline stages and the dependencies between instructions

5 Stages of the pipeline:

1. Instructional Fetch (IF)
2. Instructional Decode (ID).
3. execute (EX)
4. Memory Access (MEM)
5. Write Back (WB).

clock cycle	unit 1	unit 2.
cycle 1	IF: Load R1, @2000	IF: add R1, @2004
cycle 2	ID: Load R1, @2000	ID: add R1, @2004
cycle 3	EX: Load R1, @2000	EX, add R1, @2004
cycle 4	MEM: Load R1, @2000	MEM: add R1, @2004
cycle 5	WB: Load R1, @2000	WB: add R1, @2004
cycle 6	IF: add R1, @2008	IF: add R1, @200C
cycle 7	ID: add R1, @2008	ID: add R1, @200C
cycle 8	EX: add R1, @2008	EX: add R1, @200C
cycle 9	MEM: add R1, @2008	MEM: add R1, @200C
cycle 10	WB: add R1, @2008	WB: add R1, @200C
cycle 11	IF: store R1, @3000	
cycle 12	ID: store R1, @3000	
cycle 13	EX: store R1, @3000	
cycle 14	MEM: store R1, @3000	
cycle 15	WB: store R1, @3000	

b) Write a C-program using openMP to find the value of 'Pi'.

```
#include<stdio.h>
#include <stdlib.h>
#include <omp.h>
#define NUM_SAMPLES 1000000

int main(){
    int i, num_threads;
    double x, y, pi;
    int count=0;
    omp_set_num_threads(4);

    #pragma omp parallel private (x,y)
            reduction(+:count)

    {
    int thread_id = omp_get_thread_num();
    num_thread = omp_get_num_threads();
    unsigned int seed = 12345 + thread_id;
    #pragma ~~for~~ omp pragma for
    for(i=0; i<NUM_SAMPLES; ++){
        x=(double)rand_r(&seed)/RAND_MAX;
        y= (double)rand_r(&seed)/RAND_MAX;
        if(x** x + y* y <= 1.0){
            count++;
        }
    }
    pi = 4.0 * (double) count/(double)(NUM_SAMPLES
                    *numthreads);

    return 0;
}
```

Q2. country = "India" and Batting Type = "Right Handed" and Year of Retirement = "2012" and Highest Score = 200

Batsman ID	country
SRT	India
SCG	India
VS	India

India

Batsman ID	Batting Type
SRT	Right
RTP	Right
JHK	Right
MEW	Right
VS	Right

Right Handed

India & Right Handed

Batsman ID	country	Batting Type
SRT	India	Right
VS	India	Right

Batsman ID	Year of Retire
SRT	2012
RTP	2012
VS	2012
JHC	2012

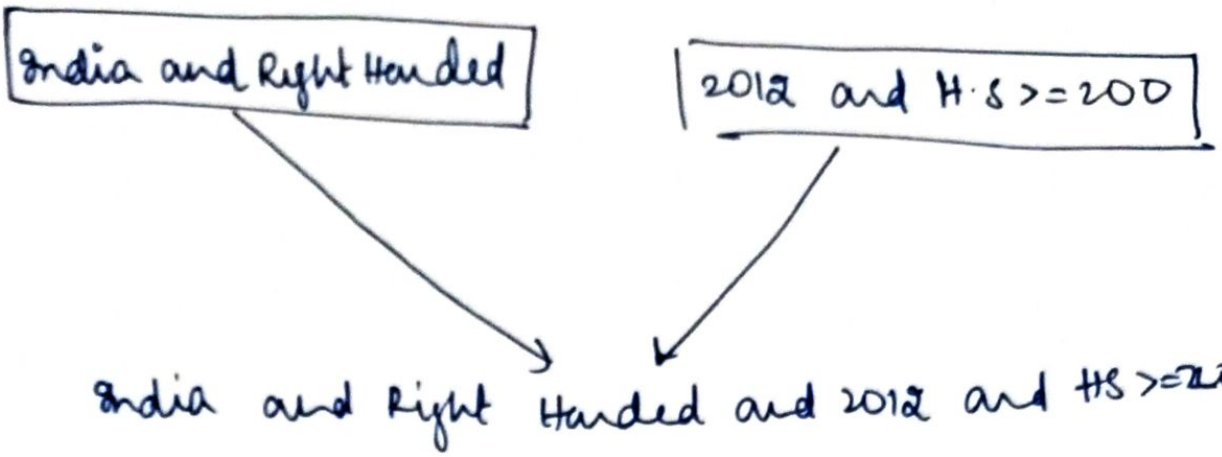
2012

Batsman ID	H·S
SRT	200
VS	264
MGB	208

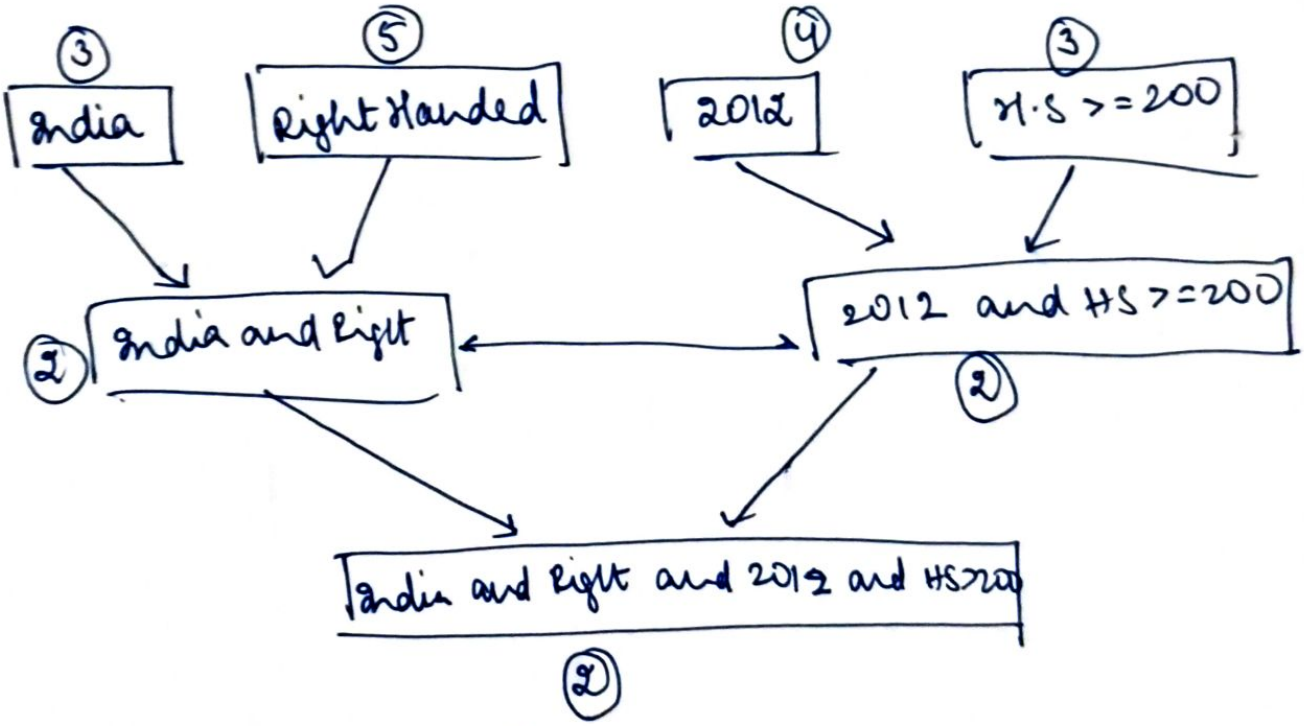
HS >= 200

2012 & HS >= 200

Batsman ID	Year of Retire	HS
SRT	2012	200
VS	2012	264



BatsmanID	Country	Retirement Year	Batting Type	HS
SRT	India	2012	Right	200
VS	India	2012	Right	264



Critical Path = 5+2+2 = 9

Total Amt of work = 21

Avg. deg of concurrency = 21/9 = 2.33

Q3.

Sol": Multithreading and prefetching do not solve all the problems related to memory system performance.

Multithreading is a technique that allows multiple threading of execution to run on the same processor core. This can help to improve performance by overlapping the execution of instructions from different threads. However, multithreading can only improve performance if the threads are independent of each other.

Prefetching is a technique that predicts which memory address will be accessed next and fetches those addresses into cache before they are actually needed. This can help into to improve performance by reducing the number of cache misses. However, prefetching can also hurt performance if it predicts incorrectly. If prefetching predicts incorrectly.

Examples:

→ memory bandwidth limitations:
The memory bandwidth is the rate at which data can be transferred between the memory and the processor. If the memory bandwidth is too low.

→ cache misses:
A cache miss occurs when data that the processor needs to is not in the cache. Cache misses can occur even if multithreading and prefetching are used.

→ contention for resources:
When multiple threads are running on same processor, they may ~~be~~ compete for resources such as processor's registers and memory controller.

Q3. c. Write a code OpenMP to make the loop iterations independent to safely execute in any order without loop carried dependency.

Soln:

```
#include <stdio.h>
#include <omp.h>

int main() {
    int n = 100;
    int sum = 0;
    #pragma omp parallel for
    {
        for (int i=0; i<n; i++){
            int result = i * i;
            #pragma omp critical
            sum += result;
            printf("Thread %d: Iteration %d result:
                %d\n", omp_get_thread_num(),i
        }
    printf("sum of results: %d\n", sum);
    return 0;
    }
}
```