P2: MIPS benchmark generation for final project

Submit Assignment

Due Monday by 11:59pm **Points** 10 **Submitting** a file upload
File Types asm **Available** after Apr 13 at 11:59pm

As part of your preparation for the final project, you will be constructing your own version of one of the two benchmarks that will run on your final project implementation (post-due-date correct standardized versions will be released for both benchmarks).

Writing these benchmarks yourself in MIPS will help you understand their behavior on the final project processor and ease debugging.

Benchmark 1 (10 points):

Naive matrix multiplication.

You will write a function, NMM, that takes 3 pointers A, B, C and 1 size parameter N and performs naive (i.e. $O(N^3)$) matrix multiplication on the NxN matrices, stored in row-major order, specified by starting locations A and B ,and stores the result into the location specified by C. All matrix elements will be 32-bit unsigned integers.

Direct C implementation operating on pointers to row-major projected 2D arrays (for clarity)

```
NMM(unsigned int * A, unsigned int *B, unsigned int * C, unsigned int N){
unsigned int i, j, k;
  for (i=0; i <N; i++){
    for (j=0; j<N; j++){
      *(C+(i*N)+j)=0; //C[i][j]=0 -- note that in C pointer arithmetic, the multiplication
of i and j by size=4 is implicit
      for (k=0; k<N; k++){
        *(C+(i*N)+j) += *(A+(i*N)+k) * *(B+(k*N)+j); //C[i][j] += A[i][k]*B[k][j] -- note
that in C pointer arithmetic, the multiplication of i, j, and k by size=4 is implicit
      }
    }
  }
```

```
}
```

This benchmark will stress the memory system in readily analyzable ways and has extremely easy to predict branch behavior.

Benchmark 2 (5 pts extra credit if correct):

Breadth-first search.

You will write a function, BFS [short for "breadth-first-search"], that takes two arguments:

1) the memory location of a graph node in an undirected graph with the following structure: (ID, depth, visited, N=number of edges, ptr-to-neighbor-node-1, ptr-to-neighbor-node-2, ... , ptr-to-neighbor-node-N)

and

2) a pointer to a pre-allocated memory region for implementing a FIFO queue (needed by the BFS algorithm - you will need to implement the queue, but you will not need to allocate memory).

The BFS algorithm is described below:

Set initial node depth to 0.

Put initial node in queue

while queue is not empty:

    current-node = remove head of queue.

    current-node.visited = true

    for node in current-node.neighbors:

        if not node.visited:

node.depth = current-node.depth + 1

put node in queue