# EI338 Computer System Engineering Homework 8

Zhou Litao 518030910407 F1803016

November 10, 2020, Fall Semester

**Exercise 1** Please explain the differences between structural hazards, data hazards, control hazards.

*Solution.*

1. Structural hazards arise from resource conflicts when the hardware cannot support all possible combinations of instructions simultaneously in overlapped execution. For example, a register can't be read and written at the same time.
2. Data hazards arise when an instruction depends on the results of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline. For example, when pipelining, read after write hazards can happen, when the data is not actually written back into the register but is requested by the next read instruction.
3. Control hazards arise from the pipelining of branches and other instructions that change the PC. For example, when a branching instruction modifies the PC, the following instructions after the branching instruction may have been loaded into the pipeline execution, which need to be abandoned.

□

**Exercise 2** Identify all of the data dependences in the following code. Which dependences are data hazards that will be resolved via forwarding?

```
ADD r2,r5,r4
ADD r4,r2,r5
SW r5,100(r2)
ADD r3,r2,r4
```

*Solution.* Data Dependence:

1. `ADD r2,r5,r4` and `ADD r4,r2,r5` on `r2`
2. `ADD r2,r5,r4` and `SW r5,100(r2)` on `r2`
3. `ADD r2,r5,r4` and `ADD r3,r2,r4` on `r2`
4. `ADD r4,r2,r5` and `ADD r3,r2,r4` on `r4`

(1), (2) and (4) can be solved by forwarding ALU results. (3) will not cause data hazards in MIPS pipeline. □

**Exercise 3**

Please write a program in OpenMP, to compute the sum of a vector. Requirements are as follows.

- Serial implement function of vector sum and parallel function of vector sum should be included.
- Use the time of serial implement function as baseline, and test the time acceleration ratio of arrays in different length under parallel function.

- Code, result should be shown in this assignment.

*Solution.*
The implementation C++ code is shown below. A running example can be found in Figure 1

□

```
#include<omp.h>
#include<iostream>
#include<fstream>

using namespace std;

int* generate_array(int len, int randl, int randr){
    int* gen = new int [len];
    for (int i=0; i<len; ++i){
        gen[i] = rand() % (randr - randl + 1) + randl;
    }
    return gen;
}

int serial_sum(int* arr, int len){
    int sum = 0;
    for (int i=0; i<len; ++i){
        sum += arr[i];
    }
    return sum;
}

int parallel_sum(int* arr, int len){
    int sum = 0;
    # pragma omp parallel for num_threads(4) reduction(+:sum)
    for (int i=0; i<len; ++i){
        sum += arr[i];
    }
    return sum;
}

int main(int argc, char* argv[]){
    int m = 1000000, l = 0, r = 5;
    // generate a random array with 1000 elements ranging from 0 to 4;
    int* arr = generate_array(m,l,r);

    double t1,t2,t3;
    t1 = omp_get_wtime();
    int sum_res1 = serial_sum(arr, m);
    t2 = omp_get_wtime();
    int sum_res2 = parallel_sum(arr, m);
    t3 = omp_get_wtime();
    cout << "Serial Time: " << t2 - t1 << "\tResult: " << sum_res1 << endl;
    cout << "Parallel Time: " << t3 - t2 << "\tResult: " << sum_res2 << endl;
    delete [] arr;
    return 0;
}
```

| **Array Size** | 10 | 100 | 1000 | $10^4$ | $10^5$ | $10^6$ | $10^7$ |
|---|---|---|---|---|---|---|---|
| SpeedUp Ratio | 0.00086 | 0.00359 | 0.02345 | 0.13191 | 1.19131 | 1.75058 | 3.99408 |

Table 1: SpeedUp Ratio VS Array Size (4 threads)

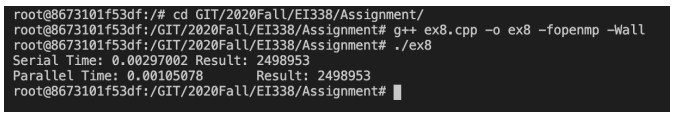


Figure 1: A running example of the openMP program