Homework 1

Due: in-class, September 25, 2019

Problem #1 (15%)

A 40-Mhz embedded processor was used to execute a benchmark program with the following instruction mix and clock cycle counts:

Instruction Type	Instruction Count	Clock Cycle Count
Integer Arithmetic	45000	1
Data Transfer	32000	2
Floating Point	15000	2
Control Transfer	8000	2

Determine the effective CPI, MIPS rate, and the execution time for this program.

Problem #2 (15%)

Let α be the percentage of a program code which can be executed simultaneously by *n* processors in a computer system. Assume that the remaining code must be executed sequentially by a single processor. Each processor has an execution rate of *x* MIPS, and all the processors are assumed equally capable.

(a) Derive an expression for the effective MIPS rate when using the system for exclusive execution of this program, in terms of the parameters *n*, α, and *x*.

(b) If n =16, and x = 4 MIPS, determine the values of α which will yield a system performance of 40 MIPS.

Problem #3 (20%)

Given the following assembly language code. Exploit the maximum degree of parallelism among the 16 instructions, assuming no resource conflicts and multiple functional units are available simultaneously. For simplicity, no pipelining is assumed. All instructions take one machine cycle to execute. Ignore all other overhead.

```
1:    Load R1, A              /R1 ← Mem(A) /
2:    Load R2, B              /R2 ← Mem(B)  /
3:    Mul R3, R1, R2          /R3 ← (R1) × (R2) /
4:    Load R4, D              /R4 ← Mem(D) /
5:    Mul R5, R1, R4          /R5 ← (R1) × (R4)/
6:    Add R6, R3, R5          /R6 ← (R3) + (R5)/
7:    Store X, R6             /Mem(X) ← (R6)/
8:    Load R7, C              /R7 ← Mem(C)/
9:    Mul R8, R7, R4          /R8 ← R7 × R4/
10:   Load R9, E              /R9 ← Mem(E)/
11:   Add R10, R8, R9         /R10 ← R8 + R9/
12:   Store Y, R10            /Mem(Y) ← (R10)/
13:   Add R11, R6, R10        /R11 ← (R6) + (R10)/
14:   Store U, R11            /Mem(U) ← (R11)/
15:   Sub R12, R6, R10        /R12 ← (R6) – (R10)/
16:   Store V, R12            /Mem(V) ← (R12)
```

(a) Draw a program graph with 16 nodes to show the flow relationships among the 16 instructions.

(b) Consider the use of a three-issue superscalar processor to execute this program fragment in minimum time. The processor can issue one memory-access instruction (Load or Store but not both), one Add/Sub instruction, and one Mul (multiply) instruction per cycle. The Add unit, Load/Store unit, and the Multiply unit can be used simultaneously if there is no data dependence.

Problem #5 (10%)

You arrive at your new internship at MSRA and on the first day you are asked to improve the performance of the following single-threaded program on a computer with one core running at 2 Ghz that can perform one math operation per clock. **The system has a memory bandwidth of 8 GB/sec, and 0 memory latency.**

```
const int N = 10000000; // very large

void slow_code(float *x, float* y) {
float tmp[N];
for (int i=0; i<N; i++)
      tmp[i] = 2.0 * x[i]; // 1 math instruction

for (int i=0; i<N; i++);
      y[i] = tmp[i] + 2.0; // 1 math instruction
}
```

Your boss asks you to improve the performance of the code by 2X. Your friend looks at the code, and says, "This is an easily parallelizable program. Let's parallelize the two loops and buy a better CPU that has two cores." You look at the code and say "I think there is a way to get a 2X speedup on our current CPU just by changing the code." Please rewrite the program to get a 2X speedup on the current CPU. Explain why this change yields a speedup.

Problem #6 (40%)

Consider the following sequence of 10 instructions. (2 memory operations, 8 arithmetic ops)

```
1. LD R0 <- [R3] // load from address given by R3
2. ADD R0 <- R0, R0
3. MUL R0 <- R0, R0
4. MUL R0 <- R0, R0
5. ADD R1 <- R0, R0
6. MUL R2 <- R0, R0
7. ADD R0 <- R0, R1
8. ADD R1 <- R1, R2
9. ADD R0 <- R0, R1
10. ST [R3] <- R0 // store to address given by R3
```

(a) Consider running this instruction sequence on a **single-core, but superscalar processor that can execute up to two independent instructions per clock.** Assume that all ten instructions (including loads and stores) complete in a single single cycle (latency = 1 clock). What is the **speedup** observed by running this instruction stream on the 2-way superscalar processor compared to a single-core processor that can only issue one instruction per clock (no superscalar)? **(Strong Hint: drawing the graph of instruction dependencies might be helpful.)**

(b) What is the speedup of a superscalar processor that is able to issue **three instructions per clock compared to a non-superscalar processor that completes one instruction per clock?**

(c) Now consider a loop where the 10-instruction sequence from part A is the body of a loop. Assume the loop is parallelized using two threads as shown below. Thread 0 computes the first N/2 iterations, and thread 1 computes the second N/2 iterations.

```
void runme(float* x, int start, int end) {

for (int i=start; i<end; i++) {
       // let the value of R3 be the address of x[i]
       // the 10-instruction sequence from part A goes here.
       }
}

float x[1000];
       // initialize values of x here...
std::thread t0(runme, x, 0, 500);
std::thread t1(runme, x, 500, 1000);
```

The two-threaded program is run on a **single-core processor with support for two hardware threads (execution contexts)**. Assume the processor works as follows: each clock it ALWAYS switches which thread it can draw instructions from. It runs up to two independent instructions from ONLY THAT ONE THREAD (if available), then switches to the next thread in the next clock. What is the speedup of this processor compared to the **2-way superscalar, single-threaded processor from part A?** Why?

(d) Now consider a small modification to the single core, multi-threaded processor from the previous problem: **each clock the single-core processor can choose to execute any mixture of up to two independent instructions from (if needed) both hardware threads** (not just one thread like in part (c). Given this new design, when running the code from part C, what is the expected speedup compared to **a single-core non-superscalar processor that completes one instruction per clock?**

(e) Now consider a situation where the load operation has a latency of 20 cycles. (There is no latency to the store, it is still one cycle.) Consider running the two-threaded program on the singlecore, **2-threaded processor that is only capable of issuing at most one instruction from one thread per clock**. What percentage of peak execution capability does the code achieve?