Problem M10.1: Multithreading

This problem evaluates the effectiveness of multithreading using a simple database benchmark. The benchmark searches for an entry in a linked list built from the following structure, which contains a key, a pointer to the next node in the linked list, and a pointer to the data entry.

```
struct node { int key;
     struct node
     *next; struct
     data *ptr;
}
```

The following MIPS code shows the core of the benchmark, which traverses the linked list and finds an entry with a particular key. Assume MIPS has no delay slots.

```
      ;
      ; R1: a pointer to the linked list
      ; R2: the key to find
      ;
loop: LW       R3, 0(R1)  ; load a key
      LW       R4, 4(R1)  ; load the next pointer
      SEQ      R3, R3, R2 ; set R3 if R3 == R2
      BNEZ     R3, End    ; found the entry
      ADD      R1, R0, R4
      BNEZ     R1, Loop   ; check the next node
End:
      ; R1 contains a pointer to the matching entry or zero
      ; if not found
```

We run this benchmark on a single-issue in-order processor. The processor can fetch and issue (dispatch) one instruction per cycle. If an instruction cannot be issued due to a data dependency, the processor stalls. Integer instructions take one cycle to execute and the result can be used in the next cycle. For example, if `SEQ` is executed in cycle 1, `BNEZ` can be executed in cycle 2. We also assume that the processor has a perfect branch predictor with no penalty for both taken and not-taken branches.

Problem 10.1.A

Assume that our system does not have a cache. Each memory operation directly accesses main memory and takes 100 CPU cycles. The load/store unit is fully pipelined, and non-blocking. After the processor issues a memory operation, it can continue executing instructions until it reaches an instruction that is dependent on an outstanding memory operation. How many cycles does it take to execute one iteration of the loop in steady state?

Problem M10.1.B

Now we add zero-overhead multithreading to our pipeline. A processor executes multiple threads, each of which performs an independent search. Hardware mechanisms schedule a thread to execute each cycle.

In our first implementation, the processor switches to a different thread every cycle using fixed round robin scheduling (similar to CDC 6600 PPUs). Each of the N threads executes one instruction every N cycles. What is the **minimum** number of threads that we need to fully utilize the processor, i.e., execute one instruction per cycle?

Problem M10.1.C

How does multithreading affect throughput (number of keys the processor can find within a given time) and latency (time the processor takes to find an entry with a specific key)? Assume the processor switches to a different thread every cycle and is fully utilized. Check the correct boxes.

	Throughput	**Latency**
Better		
Same		
Worse		

Problem M10.1.D

We change the processor to only switch to a different thread when an instruction cannot execute due to data dependency. What is the minimum number of threads to fully utilize the processor now? Note that the processor issues instructions in-order in each thread.

Problem M10.2: Multithreaded architectures

The program we will use is listed below. (In all questions, you should assume that arrays **`A`**, **`B`** and **`C`** do not overlap in memory.)

C code

```
for (i=0; i<328; i++) {
     A[i] = A[i] * B[i];
     C[i] = C[i] + A[i];
}
```

In this problem, we will analyze the performance of our program on a multi-threaded architecture. Our machine is a single-issue, in-order processor. It switches to a different thread every cycle using fixed round robin scheduling. Each of the N threads executes one instruction every N cycles. We allocate the code to the threads such that every thread executes every Nth iteration of the original C code (each thread increments **`i`** by N).

Integer instructions take 1 cycle to execute, floating point instructions take 4 cycles and memory instructions take 3 cycles. All execution units are fully pipelined. If an instruction cannot issue because its data is not yet available, it inserts a bubble into the pipeline, and retries after N cycles.

Below is our program in assembly code for this machine.

Assembly code

```
loop: ld   f1, 0(r1)   ; f1 = A[i]
      ld   f2, 0(r2)   ; f2 = B[i]
      fmul f4, f2, f1  ; f4 = f1 * f2
      st   f4, 0(r1)   ; A[i] = f4
      ld   f3, 0(r3)   ; f3 = C[i]
      fadd f5, f4, f3  ; f5 = f4 + f3
      st   f5, 0(r3)   ; C[i] = f5
      add  r1, r1, 4
      add  r2, r2, 4
      add  r3, r3, 4
      add  r4, r4, -1
      bnez r4, loop    ; loop
```

Problem M10.2.A

What is the minimum number of threads this machine needs to remain fully utilized issuing an instruction every cycle for our program? Explain.

Problem M10.2.B

What will be the peak performance in flops/cycle for this program? Explain briefly.

Problem M10.2.C

Can we reach peak performance running this program using fewer threads by rearranging the instructions? Explain briefly.

Problem M10.3: Multithreading

Consider a single-issue in-order multithreading processor that is similar to the one described in Problem M3.8.

Each cycle, the processor can fetch and issue one instruction that performs any of the following operations.

- **load/store: 12-cycle latency (fully pipelined)**
- **integer add: 1-cycle latency**
- **floating-point add: 5-cycle latency (fully pipelined)**
- **branch: no delay slots, 1-cycle latency**

The processor **does not have a cache**. Each memory operation directly accesses main memory. If an instruction cannot be issued due to a data dependency, the processor stalls. We also assume that the processor has a perfect branch predictor with no penalty for both taken and not-taken branches.

You job is to analyze the processor utilizations for the following two thread-switching implementations.

Fixed Switching: the processor switches to a different thread every cycle using fixed roundrobin scheduling. Each of the N threads executes an instruction every N cycles.

Data-dependent Switching: the processor only switches to a different thread when an instruction cannot execute due to a data dependency.

Each thread executes the following MIPS code.

```
loop: ld   F2, 0(R1)      ; load data into F2
      addi R1, R1, 4      ; bump source pointer
      fadd F3, F3, F2     ; F3 = F3 + F2
      bne  F2, F4, loop   ; continue if F2 != F4
```

Problem M10.3.A

What is the minimum number of threads that we need to fully utilize the processor for each implementation?

Fixed Switching: _____ Thread(s)

Data-dependent Switching: _____ Thread(s)

Problem M10.3.B

What is the minimum number of threads that we need to fully utilize the processor for each implementation if we change the **load/store latency to 1-cycle (but keep the 5-cycle floatingpoint add)**?

Fixed Switching: _____ Thread(s)

Data-dependent Switching: _____ Thread(s)

Problem M10.3.C

Consider a **Simultaneous Multithreading (SMT)** machine with limited hardware resources. Circle the following hardware constraints that can limit the total number of threads that the machine can support. For the item(s) that you circle, briefly describe the minimum requirement to support N threads.

(A) Number of Functional Units

(B) Number of Physical Registers

(C) Data Cache Size

(D) Data Cache Associatively

Problem M10.4: Multithreading (Spring 2015 Quiz 2, Part D)

Consider the following instruction sequence.

```
       addi   r3, r0, 256
loop:  lw     f1, r1, #0
       lw     f2, r2, #0
       mul    f3, f1, f2
       sw     f3, r2, #0
       addi   r1, r1, #4
       addi   r2, r2, #4
       addi   r3, r3, #-1
       bnez   r3, loop
```

Assume that memory operations take 4 cycles (i.e., if instruction I1 starts execution at cycle N, then instructions that depend on the result of I1 can only start execution at or after cycle N+4); multiply instructions take 6 cycles; and all other operations take 1 cycle. Assume the multiplier and memory are pipelined (i.e., they can start a new request every cycle). Also assume perfect branch prediction.

Problem M10.4.A

Suppose the processor performs fine-grained multithreading with fixed round-robin switching: the processor switches to the next thread every cycle, and if the instruction of the next thread is not ready, it inserts a bubble into the pipeline. What is the minimum number of threads required to fully utilize the processor every cycle while running this code?

Problem M10.4.B

Suppose the processor performs coarse-grained multithreading, i.e. the processor only switches to another thread when there is a L2 cache miss. Will the following three metrics increase or decrease, compared to fixed round-robin switching? Use a couple of sentences to answer the following questions.

1) Compared to fixed round-robin switching, will the **number of threads needed for the highest achievable utilization** increase or decrease? Why?

2) Compared to fixed round-robin switching, will the **highest achievable pipeline utilization** increase or decrease? Why?

3) Compared to fixed round-robin switching, will **cache hit rate** increase or decrease? Why?