Problem 1. “To Compute a Pipelined Pixel, LEEly”

Upon finishing Comp 411 Lee Hart was immediately hired by a hot new start-up company, Pixzilla, which specializes in the design of high-end graphics processing units. On his first day of work his supervisor and company founder, Zeeb Uffer, explains to him that virtually all computer graphics functions can be computed using just one data path called a Linear Expression Evaluator, or LEE for short. A LEE simply computes the following linear function:

\[ \text{LEE}(x, y) = Ax + By + C. \]

The entire process of rendering a triangle can be reduced to the evaluation of a series of LEEs. For instance, whether or not a pixel \((x, y)\) lies inside of a triangle can be determined using 3 LEEs, one for each edge. In this case, the “zero-set” \((Ax + By + C = 0)\) describes the equation of an edge. The signs of the \(A, B,\) and \(C\) coefficients are chosen so the linear expression is positive inside of the triangle, and negative outside. Thus, a pixel inside of a triangle will give a positive result for all three edge LEEs.

Furthermore, the color of the triangle can be interpolated within the triangle using a single LEE for each primary color (i.e. \(\text{red}(x, y) = Ax + By + C\)). If the color at each vertex is known, the values of \(A, B,\) and \(C\) are determined by solving a linear system using the given values of the colors and the coordinates of the three vertices. Likewise, the depth at each pixel inside a triangle can be interpolated using a LEE when the depth at each vertex is given.

Currently Pixzilla uses the following implementation of a LEE:

![Diagram of LEE implementation]

Lee’s job is to improve the performance of this LEE, using the pipelining skills learned in Comp 411.
(A) Assuming that the propagation delay of the multipliers shown is 40 pS, and the propagation delay of the adders shown is 10 pS, what is the propagation delay and throughput of the initial LEE implementation?

(B) If the LEE implementation shown is pipelined using registers with 5 pS setup time and a 5 pS propagation delay, what is the maximum throughput that can be achieved? What is the minimum latency that achieves this maximum throughput?

(C) Lee thumbs through the schematic of the Pixzilla LEE implementation and realizes that the multiplier shown in the diagram above is actually implemented using four smaller multipliers as shown below:

Lee discovers that the small multipliers have a propagation delay of 25 pS, the small adder (labeled +a) has a propagation delay of 5 pS, the larger adder (labeled +b) has a propagation delay of 10 pS and registers have the same timing as in part (B). With his newfound knowledge, what is the maximum throughput that can be achieved for this LEE implementation? What is the minimum latency that achieves this maximum throughput?

(D) Given that each rendered pixel requires 7 LEE evaluations, and that the average triangle turns on only 30 pixels, what is the maximum possible number of triangles per second that Pixzilla’s accelerator can render?

Problem 2. “Parallel Pipelined Pixels (a.k.a Lee’s LEE Lulu)”

After working at Pixzilla for a couple of days, Lee develops a greater understanding of the graphics accelerator in which the LEEs are used. He discovers that the coefficient values of the LEE are generally held constant while the values of x and y are scanned through the bounding box of the triangle being rendered. Lee decides to increase the parallelism of the LEE, but he is reluctant to add any more multipliers, because of their expense. Suddenly it dawns on Lee that it is possible to build a parallel LEE that uses no multipliers. Consider the following functional block diagram (Note: each box contains the circuit indicated on the left and a schematic representation of the entire circuit is shown on the right):
(A) Determine the expression computed at each of the labeled outputs (O₀ to O₇).

(B) Lee immediately describes his discovery to Zeeb. Zeeb is confused about two aspects of Lee’s design. First, he does not understand how the divide-by-2 circuit is implemented. Lee exclaims, “It requires no gates, only wires.” Can you explain what Lee means? Next Zeeb questions how outputs can be generated for any arbitrary value of x. Lee explains that any values can be accommodated by adjusting the value of C. Can you explain how?

Lee designs the following parallel LEE implementation using the circuit shown above:

(C) Given that each adder in Lee’s circuit has a propagation delay of 10 pS, what is the total propagation delay of Lee’s parallel LEE? What is its throughput (in terms of LEEs/sec)? What is the total number of adders used in Lee’s parallel LEE?

(D) Lee then sets out to pipeline his parallel LEE implementation using registers with 5 pS setup time and a 5 pS propagation delay. What is the maximum throughput (in LEEs/sec) that can be achieved? What is the minimum latency that achieves this maximum throughput?
The disadvantage of Lee’s parallel LEE implementation is that linear expressions are computed for an entire block of pixels even though many of these pixels might lie outside of the specified triangle. Suppose that on average about 40% of the LEEs evaluated by Lee’s parallel implementation are utilized. Remember that each rendered pixel requires 7 LEE evaluations, and that the average triangle turns on only 30 pixels. What is the maximum possible number of triangles per second that an accelerator based on Lee’s pipelined-parallel LEE can render?

Problem 3. Delayed Decisions

Many modern instructions set architectures include special conditional instructions designed to avoid branch delays and to pipeline stalls related to determining a branch target. Consider the following proposed extension to the miniMIPS ISA.

```
abnz rt,rs,label
```

Add the contents of register rs to those of register rt, if the result is not zero branch to label.

```
if (Reg[rt] + Reg[rs] != 0) {
   PC ← PC + 4 + 4*sign_extend(imm16);
}
Reg[rt] ← Reg[rt] + Reg[rs]
```

(A) What instruction format would the abnz instruction use?

(B) How should each miniMIPS control signal be set to implement the abnz instruction (assume the unpipelined miniMIPS implementation) HINT: you should specify PCSEL as a function of the ALU’s Z flag?

(C) At first glance it might seem that subtracting Reg[rs] from Reg[rt] is a more natural instruction choice. Explain why this change would require datapath modifications.

Consider the following two implementations of the procedure int sum(int N). The first uses only standard MIPS instructions while the second takes advantage of the abnz instruction:

```
sum1: addu $sp,$sp,-24 move $v0, $0 loop: add $v0,$v0,$a0 addi $a0,$a0,-1 bne $a0,$0,loop addu $sp,$sp,24 j $31

sum2: addu $sp,$sp,-24 move $v0, $0 addi $t0,$0,-1 loop: add $v0,$v0,$a0 abnz $a0,$t0,loop addu $sp,$sp,24 j $31
```

(D) For what values of the argument N is sum2 is at least 25% faster than sum1?

Despite the apparent advantages of the abnz instruction (it requires no additional H/W and it improves the performance of some loops), there are still significant reasons for not including it.

(E) One problem with the abnz instruction is that it is difficult to pipeline. At what stage in the miniMIPS 5-stage pipeline is the branch decision made for the abnz instruction? How many delay slots would a straightforward implementation of it require? Describe the additional logic that would be required to compute an early branch decision in the
Register-Fetch pipeline stage for the abnz instruction. How does the complexity, and likely propagation delay, of the early branch-decision hardware required for the abnz instruction compare to that of the bne and beq instructions of the standard MIPS ISA.

Another difficulty associated with special-purpose branching instructions is that it is often difficult for compilers to take advantage of them. Consider the following C-code fragment:

```c
int sum = 0;
for (int i = 0; i < N; i = i + 1)
    sum = sum + x[i];
```

(F) Write a MIPS assembly language code fragment for the loop given above using the standard MIPS branch instructions, and then recode your fragment incorporating the abnz instruction. Comment on the coding and conceptual difficulties associated with incorporating the abnz instruction in this loop (Note: In order to support debugging it is required that the sum be computed in the same order as specified by the C-code).

**Problem 4. Flexible Pipes**

Bud LeVile has suggested a modification to the 5-stage miniMIPS pipeline discussed in class. Having noticed that the MEM stage is only used for load and store instructions, he proposes omitting that pipeline stage entirely whenever the memory isn’t accessed, as illustrated below:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>t</th>
<th>t+1</th>
<th>t+2</th>
<th>t+3</th>
<th>t+4</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw and sw</td>
<td>IF</td>
<td>RF</td>
<td>ALU</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>Other instrs</td>
<td>IF</td>
<td>RF</td>
<td>ALU</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bud reasons that instructions which skip the MEM stage can complete a cycle earlier, thus, allowing most programs will run as much as 20% faster! In your answers below assume that both the original and the Bud-modified pipelined implementations are fully and properly bypassed.

(A) Explain briefly to Bud why decreasing the latency of a single instruction does not necessarily have an impact on the throughput of the processor (Hint: Consider how long it would take the original pipelined miniMIPS to complete a sequence of 1000 adds. Then compare that with how long a Bud-modified miniMIPS would take to complete the same sequence).

(B) Consider a sequence of alternating lw and add instructions. Assuming that the lw instructions use different source and destination registers than the add instructions (i.e., there are no pipeline stalls introduced due to data dependencies), what is the instruction completion rate of the original, unmodified 5-stage miniMIPS pipeline?

(C) Now show how the same sequence of instructions will perform on a processor modified as Bud has suggested. Assume that the hardware will stall an instruction if it requires a pipeline stage that is currently being used by a previous instruction. For example, if two instructions both want to use the Write-Back pipeline stage in the same cycle, the instruction that started later will be forced to wait a cycle. Draw a pipeline diagram showing where the stalls need to be introduced to prevent pipe stage conflicts.

(D) Did Bud’s idea improve performance? Explain why or why not?
Problem 5. Stage Three

Suppose that the behavior of the lw and sw instructions were redefined as follows:

\[
\text{l}
\text{w}
\quad \text{rt, (rs)} \quad \text{Reg}[rt] \leftarrow \text{Mem}[\text{Reg}[rs]]
\]

Load register rt with the contents of the memory location specified register rs.

\[
\text{s}
\text{w}
\quad \text{rt, (rs)} \quad \text{Mem}[\text{Reg}[rs]] \leftarrow \text{Reg}[rt]
\]

Store the contents of register rt at the memory location specified register rs.

(A) Give instruction sequences that emulate the operation of the original lw and sw instructions as pseudoinstructions using the redefined versions. Note: Use register $sas$ to store any required intermediate values.

These ISA changes enable memory accesses and ALU operations to be overlapped in the same pipeline stage (ALU/MEM). They also allow for the construction of a meaningful 3-stage miniMIPS pipeline, whose datapath is illustrated below:

(B) Discuss where and the how many bypass paths are needed for this modified architecture. Give an example instruction sequences that exercises each bypass path.

(C) Does this modified 3-stage pipeline architecture require pipeline interlocks on load instructions? Explain why or why not.