This is a full length practice midterm exam. If you want to take it at exam pace, give yourself 75 minutes to take the entire test. Just like the real exam, each question has a point value. There are 75 points in the exam, so that you can pace yourself to average 1 point per minute (some parts will be faster, some slower).

Questions:

1. C Programming [15 points]
2. Boolean Algebra + Gates [10 points]
3. Flipflops [10 points]
4. Performance, etc... [10 points]
5. Datapaths [10 points]
6. Finite State Machines [10 points]
7. Short-answer [10 points]

This is the solution set to the practice exam. The solutions appear in blue boxes.
Question 1: C Programming [15 pts]

Given the following type-definition for a binary search tree node:

```c
struct _bst_node {
    int key;
    struct _bst_node * left;
    struct _bst_node * right;
};
typedef struct _bst_node bst_node;
```

Write the function `countBetween` which returns a count of how many items in the binary search tree have a key in between `min` and `max` (inclusive of both). Your implementation should make use of the binary-search tree’s ordering properties:

```c
int countBetween(bst_node * root, int min, int max) {
    if (root == NULL) {
        return 0;
    }
    if (root->key < min) {
        return countBetween(root->right, min, max);
    }
    if (root->key > max) {
        return countBetween(root->left, min, max);
    }
    return 1 +
        countBetween(root->left, min, max) +
        countBetween(root->right, min, max);
}
```
Question 2: Boolean Algebra + Gates [10 pts]

For the following truth table

- Write the formula for the truth table in sum-of-products form.
- Simplify the formula as much as possible.
- Draw the logic gates which correspond to your simplified formula.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Answer:

SOP: (!A&!B&C) | (!A&!B&C) | (A&!B&C) | (!A&!B&!C) | (A&!B&!C)

Simplified: !B | (!A&B&C)
Question 3: Flipflops [10 pts]
Consider the following circuit (assume all flip-flops are positive edge triggered):

Draw the waveforms (showing the value for each signal at each time) for this circuit. Assume that A and D start at 1, and B and C start at 0.
Question 4: Performance... [10 pts]

Suppose that there are 3 processor designs to choose from, with the following characteristics:

<table>
<thead>
<tr>
<th>Name</th>
<th>CPI</th>
<th>Clock Freq (MHz)</th>
<th>Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5.0</td>
<td>2000</td>
<td>40</td>
</tr>
<tr>
<td>B</td>
<td>6.0</td>
<td>3000</td>
<td>250</td>
</tr>
<tr>
<td>C</td>
<td>4.0</td>
<td>2400</td>
<td>100</td>
</tr>
</tbody>
</table>

All 3 processors execute the same ISA, so instructions/program is constant. Which processor would you select in each of the following situations? Justify your answer by stating what metric is appropriate and showing how each processor compares on that metric.

1. You need one single computer for your office, and it has to perform computationally intensive tasks quickly.

**Answer:**
Here, we want to compare performance, so MIPS is the best metric to use.
A: $2000 \times (1/5) = 400$ MIPS
B: $3000 \times (1/6) = 500$ MIPS
C: $2400 \times (1/4) = 600$ MIPS
So we would pick processor C.

2. You want a laptop chip that will maximize battery life (ignoring other factors like the screen and disk-drives, and assuming that the processor can go idle and use almost no power once it finishes running a computation).

**Answer:**
Here, we care about energy (power times time). Power is given already, and the time to execute a program is (million instructions / MIPS). Since the number of instructions is constant, energy is proportional to W/MIPS. We can look for the smallest W/MIPS, or (to make the numbers easier to work with) the largest MIPS/W:
A: $400$ MIPS / 40 W = 10 MIPS/W
B: $500$ MIPS / 250 W = 2 MIPS/W
C: $600$ MIPS / 100 W = 6 MIPS/W
So we would pick processor A.
Question 5: Datapaths [10 pts]

RISC ISAs tend to be load-store ISAs—loads and stores are the only instructions which access data memory, and that is all that they do. Suppose that we were to want to add an instruction to the MIPS ISA (and thus datapath) which violated this principle. This new (made up) instruction \texttt{ladd $rd, ind($rs), $rt} (short for load-add) has the semantics that it computes $\text{Mem}[\text{rs+int}] + \text{rt}$—that is, it computes ind + $\text{rs}$ and uses that as an address to access data memory. It then adds the result read from data memory to $\text{rt}$, and places the resulting sum in $\text{rd}$.

1. Show the datapath modifications required to the basic single-cycle MIPS data-path we have been working with in lecture:
2. Does this seem like a good idea or a bad idea from a performance standpoint (or, if “it depends”, under what conditions is it a good idea)?

**Answer:**
This addition is going to affect two components of performance: the clock period, and the number of instructions per program (Since the data-path is single-cycle, CPI is fixed at 1).

**Clock period:** The datapath is going to need to be clocked more slowly because there is another ALU to go through after the data memory access (before writing the register file). A good guess would be about 6/5 as long (5/6 as fast), since we use a stage for the ALU in a 5-stage multi-cycle datapath.

**Instructions per program:** Anytime we would normally have lw followed by add, we can turn those two instructions into a single instruction. How many lw-followed-by-adds there are is going to depend on the program, but we will only come out ahead if we can cut out more than about 1/6 of the total instructions (6/5 * 5/6 = 1). This does not seem terribly likely.

Side-note: If this were a real question, you would have a fair bit of flexibility in justifying your answer, as long as you justify it—e.g., you would not need to be quite as specific with the numbers.

3. Briefly explain how the situation would be different in a *multi-cycle* data-path.

**Answer:**
In a multi-cycle data-path, this instruction is much more appealing. Here, the clock frequency would remain the same, but the ladd instruction would take 6 cycles (compared to the load which would take 5). Whenever the compiler can find a load/add pair it can replace with ladd, it is taking 9 cycles (4 + 5) and replacing it with 6 cycles, for a clear win.
Question 6: Finite State Machines [10 pts]

Draw a state machine diagram for a finite state machine which accepts a single bit input (either 0 or 1—you can just label each edge with 0 or 1). This state machine also has a single bit of output, which is initially 0.

- Whenever the FSM receives an input sequence of 101 or 110 the output bit goes to 1.
- The output remains at 1 until the FSM receives an input sequence of 11.
- Once the FSM receives the input sequence of 11, the output bit returns to 0.
- The output now remains 0 until 101 or 110 are input again, in which case the output bit goes to 1 and the process repeats.

Label each state with the bit it outputs. Be sure to indicate your start state (with an arrow to it from nowhere).
Question 7: Short-answer [10 pts]

1. Briefly explain what is wrong with the following fragment of C code:

```c
while (curr->next != NULL) {
    if (curr->next->data == toRemove) {
        free(curr->next);
        curr->next = curr->next->next;
    } else {
        curr = curr->next;
    }
}
```

**Answer:**
The problem here is that curr→next is being free()ed, then used (curr→next→next). Once the memory is free()ed, it should not be accessed again.

2. If a single cycle datapath has a clock period of 100ns, and is split into a multi-cycle data-path with 5 stages, the clock will be slower than 20ns. Give two reasons why:

- **Answer:**
The 5 stages may not be exactly balanced (and probably won’t be). E.g., they probably will 19-21-20-21-19 rather than 20-20-20-20-20. Since the clock must be constrained by the longest stage, it would need to be 21ns rather than 20ns.

- **Answer:**
We have not just divided up the data-path, but also added some logic (namely registers—made of D-flip-flips). This additional logic takes some time to propagate through, so we must add a little more to the cycle time to account for it.