Problem 0: Warm-up (2 points)

Facebook is hiring hardware engineers. What do you think they are building?

Any answer is accepted.

Problem 1 (43 points)

Part a (5 points): A microarchitecture is predicting whether a branch is taken or not taken using a 1-bit predictor. The last five branches were: taken, taken, taken, taken, not taken. What does the branch predictor predict (choose): Taken or Not Taken?

Not Taken

Part b (5 points): In a typical Linux/Unix terminal, when you hit Ctrl + Z, which state are you putting the foreground process to (choose): Running, Stopped, Terminated?

Stopped

Part c (5 points): Cache blocking is a software-level performance optimization technique that improves what aspect of a program (choose): Locality, Parallelism, Concurrency, Security?

Locality

Part d (5 points): An application that is 90% parallelizable is executed on a single processor in 1.5 hours. If the application is allowed to run with an unlimited number of processors, what is the lower bound on its execution time?

9 mins. With an unlimited number of processors, the parallelizable part of a program would finish in no time, and the execution time is equivalent to the sequential part, which is 90 mins * 0.1 = 9 mins.

Part e (5 points): On a page fault, the operating system often loads a page from the disk into memory. How does the operating system know whether it is necessary to write the previously occupied page in the memory back to the disk? Answer in fifteen words or fewer.

Check the dirty bit in the corresponding page table entry.

Part f (5 points): What is the cycle time of a 1 GHz processor?

1 ns

Part g (5 points): What is the fundamental reason that process context switch has a much higher overhead than thread context switch in Linux? Answer in twenty words or fewer.

Threads share virtual address space while processes have separate virtual address spaces.

Part h (5 points): Suppose we have two 4-bit 2's complement numbers:

1111
1110

Does the sum of the two numbers result in an overflow?

No.

Part i (3 points): Recall that the crux of tracing-based GC algorithms such as Mark-and-sweep and Mark-sweep-compact is to start from "root" variables and then identify all the reachable variables. In the following code snippet, suppose the program just finishes executing L7, which variables are regarded as "root"? Name only those that point to variables on the heap.

```
L1:    int *p3;

L2:    int* foo(int n) {
L3:      int i, *p1;
L4:      p1 = (int *) malloc(n * sizeof(int));
L5:      for (i=0; i<n; i++)
L6:        p1[i] = i;
L7:      p3 = p1[2];
L8:      return p1;
L9:    }

L10:   void bar() {
L11:     int *p2 = foo(5);
L12:   }
```

`p1` and `p3`.