

By writing my name on this sheet, I certify that all the work on this quiz is mine alone, and that I will abide by the honor code as printed.

As a student of The University of Texas at Austin, I shall abide by the core values of the University and uphold academic integrity.

— Student Honor Code

Name:

Signature:

1	2	3	4	T

1. (3 points) Take a look at the following code segment. Suppose that it's run on two cores (their IDs being 0 and 1), both of which call `kernelMain`. Also suppose that `critical` is implemented correctly as declared in the p1 spec.

```
static int counter = 0;

void work() {
    Debug::printf("*** %d %d\n", SMP::me(), counter++);
}

void kernelMain() {
    critical(work);
}
```

- (a) (1 point) How many possible outputs are there?

- (b) (2 points) List them.

2. (2 points) Answer the two questions below about critical sections.

(a) (1 point) What is the time constraint that we place on code that runs in a critical section?

(b) (1 point) What is a good reason why we have this constraint?

3. (2 points) At a high level, what does the code in `mbr.S` accomplish in terms of the boot process?

4. (3 points) Uh oh! Dr. Gheith deleted his `atomic.h` on accident — and, just before he was set to release p1. He has to rewrite it, and he decides to start with `SpinLock`. Since he likes to challenge himself, he has decided to do it in terms of the gcc atomic built-ins directly instead of implementing a wrapper first. He has written out the following:

```
class SpinLock {
    Atomic<bool> taken;
public:
    SpinLock() : taken(false) {}

    void lock(void) {}

    void unlock(void) {}
};
```

He is staring at the gcc API, contemplating about the following built-ins:

```
// This built-in function implements an atomic load operation.
// It returns the contents of *ptr.
type __atomic_load_n (type *ptr, int memorder)

// This built-in function implements an atomic store operation.
// It writes val into *ptr.
void __atomic_store_n (type *ptr, type val, int memorder)

// This built-in function implements an atomic exchange operation.
// It writes val into *ptr, and returns the previous contents of *ptr.
type __atomic_exchange_n (type *ptr, type val, int memorder)

// This built-in function implements an atomic add-fetch operation.
// It adds val to the value stored in *ptr, and returns the new contents of *ptr.
type __atomic_add_fetch (type *ptr, type val, int memorder)
```

(a) (0.5 points) Which of the above atomic operations should he use to implement `lock`?

(b) (0.5 points) What about `unlock`?

(c) (1 point) In pseudocode, implement `lock`.

(d) (1 point) In pseudocode, implement `unlock`.