312 Lab1

Patrick Rock

September 2013

1 Problem 1

1.1 a)

Tag 1 declares an int variable. This tells the compiler to create an entry in the symbol table for the identifier. Tag 2 writes the size of an int in bits and bytes to standard out. The % characters are special formating symbols for the other printf arguments.

1.2 b)

The sizeof() function returns the number of bytes in its argument. It is not a system call, it is a C function.

1.3 e)

No, time was positive in both cases.

- Unix: 1104448641 secs
- Linux: 465588427 secs

1.4 f)

There is a change. The values are more consitant with eachother. The second values are correct. In the first trial, the double variable was overflowing and wrapping around. Long int doesn't overflow in this case.

- Unix: 1379009658 secs
- Linux: 1379009718 secs

1.5 g)

Timeval is a C struct. It is not platform specific. The defenition of timeval on any system is:

```
1 typedef struct timeval {
2   long tv_sec;
3   long tv_usec;
4 } timeval;
```

2 Problem 2

2.1 b)

I learned that on my personal machine unsigned ints and floats are the same size, 4 bytes. Different types have varying sizes. Since there are multiple types of the same size, the way bits represent information must be different for each type.

3 Problem 3

3.1 a) Requirements

1. $BELL = ER * DSBF'$
2. $BELL = ER * DC'$
3. $BELL' = DSBF * DC$
4. $DLA' = DOS' * KIC \; DLA = DOS * DLC$
5. $BA = BP * CM$

3.2 b) Truth Table

DOS	DSBF	ER	DC	KIC	DLC	BP	CM	BELL	DLA	BA
x	0	1	0	x	x	x	x	1	x	x
x	1	1	0	x	x	x	x	1	x	x
x	0	0	0	x	x	x	x	0	x	x
x	1	0	0	x	x	x	x	0	x	x
x	0	1	1	x	x	x	x	1	x	x
x	1	1	1	x	x	x	x	0	x	x
x	0	0	1	x	x	x	x	0	x	x
x	1	0	1	x	x	x	x	0	x	x
1	x	x	x	0	1	x	x	x	1	x
1	x	x	x	1	1	x	x	x	1	x
0	x	x	x	1	0	x	x	x	0	x
0	x	x	x	1	1	x	x	x	0	x
1	x	x	x	0	0	x	x	x	0	x
1	x	x	x	1	0	x	x	x	0	x
0	x	x	x	0	1	x	x	x	0	x
0	x	x	x	0	0	x	x	x	0	x
x	x	x	x	x	x	1	1	x	x	1
x	x	x	x	x	x	0	1	x	x	0
x	x	x	x	x	x	1	0	x	x	0
x	x	x	x	x	x	0	0	x	x	0

4 Problem 4

4.1 b)

I designed my original program with functions. These helper functions hide the details of bitmasking and eliminate the need for macros.

5 Problem 5

5.1 a)

The linux machine ran much faster than the unix machine. Unix has a much wider timer resolution than linux. Unix took much longer to calibrate than linux.

5.2 b)

The code for problem 4 is much slower than the code for problem 3. I implemented problem 4 using many accessor functions to simplify the low level bit operations. The extra function calls in control_action() slow down execution dramtically.

5.3 c)

Executing the command cat /proc/cpuinfo on the linux machine show us that the cpu clock speed is cpu 2999.976 MHz. Since this is near the 4GHz upper limit for cpu frequency, it is not likely that we solve the problem by throwing hardware at it.

5.4 d)

A hardware solution would be faster and cheaper than a software solution. We can design a logic circuit to map sensor values to actuator actions.

6 Problem 6

6.1 a)

The truth table for this curcuit should be identical to the truth table in section 3.2

6.2 b)

The bell is the slowest to respond, and it takes 10.2 nanoseconds. The other actuators respond in 3.4 nanoseconds. This is well within the 50 nanosecond constraint.

6.3 c)

For simple applications, a direct hardware solution is faster and cheaper than a software one. We also eliminate the need for a microprocessor.