EC413 Computer Organization
Lab 1 – C and your CPU

The purpose of Lab 1 is to review:
- The C programming language

- Basic Linux command‐line workflow

Also, to introduce some aspects of C that bring you close to the underlying processor and to demonstrate that, even with a high‐level language, your CPU can be exposed with a simple program.

Deliverables:
- Your code
- Output from run of your code
- Answers to questions

Note: NO DEMO NEEDED FOR LAB 1
LAB TO BE DONE INDIVIDUALLY

1. Compiling and running programs

- To compile, type "cc ‐o testC testC.c".
- To run, type "./testC". You should see the message "Hello world!" and some other stuff.
- Change "World" to your name, recompile and rerun.

2. Sizes of datatypes
- Write code that uses the "sizeof()" function to find the sizes of the C data types. One example (for int) is given in the sample code. Repeat for some other data types.

3. Signed versus unsigned datatypes
- Explain why uchar1 and schar1 have different output even though they were assigned the same value.
- Write code to compare uchar1 and uchar2 and print the value of the larger.
- Write code to compare schar1 and schar2 and print the larger.
- Try mixing signed and unsigned types, e.g., by comparing schar1 and uchar1. What happens?
- Now add schar1 and schar2 and print the sum. What is it? Is this what you expected? If not, then explain what happened.
- Now add uchar1 and uchar2 and print the sum. What is it? Is this what you expected? If not, then explain what happened.
- What happens when you add schar1 and uchar1?

4. Booleans.
- Print out the values for Boolean TRUE and Boolean FALSE. You might need to think about this for a minute
  - There is no Boolean type in C, but it is possible to assign the result of a Boolean operation (e.g., && or ||) to another variable, or to print it out directly.

- How big is the internal data type for a “Boolean”? Hint: Use sizeof() on a Boolean expression.
- Write code that shows that the “&” and the “&&” operands can behave differently.
- Write code that shows that the “~” and the “!” operands can behave differently.

5. Shifts

- Write code to shift *shift_char* left and right by some number of places. What values do you get?
- What happens when you shift it left more than 3 places? Is this what you expect? Explain what happened.
- What happens when you shift it left more than 7 places?

6. Pointer basics

- Shown is the example from class. Does it give the correct answer (0 0 10 0 10 40 0)? If yes, then continue. If not, then this is likely because the C I/O interface does not interact predictably with the operating system. If necessary, correct this example by printing each value separately (in its own statement).
- Change the elements being printed (a[0], *(ip), etc.) so that the output is now (10 10 20 10 20 50 10). Try to do this with minimal changes, such as a[0] to a[1].
- Write code to determine the size of the integer pointer data type.
- What happens when you print the pointer values themselves? You might want to use %x instead of %d to get the values in hex (why?). Why is the difference between ip and ip+1 not 1?

7. Programming with pointers

- Write two code fragments that reverse a[ ]. For the first, use conventional array indices; put the result into b[ ]. For the second, use pointers and put the result into array c[ ]. Print the inputs and outputs.

8. Strings

- Write code that prints the ASCII values of the characters in the string (not the string itself).
- Print the value of the byte after the last character. What is it? Why?
- Add integer 1 to each character in the string and print. What do you get?
- Add integer 60 to the byte following the last character in the string and print the entire string (as in the given code). What happens?

9. Address Calculations (of array indices)

- Two code fragments from the lecture are given. Convince yourself that the address calculations are the same in both cases by printing out the addresses of the output array as they are being computed (inside the transfer loop).