CPE 315-01 Computer Architecture

Lab 1

Submitted 1/12/2018

Samuel Sachnoff and Michael Silin

Professor David Retz

Cal Poly San Luis Obispo

• Introduction

In this lab we learned about data item sizes, byte ordering, basic binary arithmetic, carry and overflow. The purpose of this lab is to become familiar with binary data types using C.

• Functional Requirements

We accomplished this by first measuring the size of a byte, short int, int, and long int and then creating a simple program to do the rest of the assignment. It's worth mentioning that the size of these values depends on the processor, operating system, and compiler options.

• Approach

Part 1, we measured the size of the data types by using the sizeof( ) function.

Part 2

The first step of our program was to define a test array. We then measured the adjacent values of the array by defining our pointer and using the printf function. We performed this step for different data sizes(short, int, long) using little endian format, then followed up by writing our expectations of the values in big endian format.

Part 3

We wrote a few lines of code which allowed us to print unsigned hexadecimal additions via just adding 2 unsigned chars and returning that result.

Part 4

We added 2 unsigned characters. If that result was less than either one of the operands, we knew a carry occurred. Additionally, to be on the safe side we truncated any bits after the first two bytes.

Part 5

In this part we determine if any overflow occurred when adding two signed chars. If the two operands had the same sign, but the result had a different sign, overflow occurred.

Part 6

In this part, we allocated memory of specified by the size parameter in the given function prototype to store the resulting answer. Furthermore, we stored the hex string to add in 2 arrays of unsigned characters. For the provided test cases they were both of size 5. Then we added 2 unsigned characters and determined if there was any carry when doing so. We then stored the result and operand pointers.

• Listing of program – Source code, with Comments explaining the operation of the code. Each function must include a header that describes calling conventions, assumptions, and return parameters.

```
/* prints out an array of size 16 of unsigned chars as if it were run on a little
 * endian machine. This function has no return value
 */
static void printArray2a(unsigned char testArray[]);

/* prints out an array of size 16 of unsigned chars as if it were run on a big
 * endian machine. This function has no return value
 */
static void printArray2e(unsigned char testArray[]);

/* this function prints out an array in the following formats: unsigned char, short,
int, long.
 * It also prints that array out starting from low byte order as if it were run on both
big and little
 * endian machines. This function has no return value. Problems 2a- 2e print the
array out as if this was run on little endian machine.
 * Problems 2a and 2e print the array out as unsigned chars. Problems 2b and 2f
print the array out as shorts. Problems
 * 2c and 2g print the array out as ints. Problems 2d and 2h print the array out as
longs.
 */
void runProblem2();

/* prints out the size of a char, short, int, and long on a given machine
 * this function has no return value
 */
void runProblem1();

* adds 2 unsigned chars. Displays the result as an unsigned char. returns an
unsigned char for use
 * later problems if necessary.
 */
static unsigned char addUnsignedChars(unsigned char a, unsigned char b);
```

```
/* this function makes function calls to addUnsignedChars. has no return value.
 */
void runProblem3()

/* This function determines if a carry occurred when adding 2 values.
 * This function returns an unsigned char as return value may be needed
 * in later problems.
 */
static unsigned char determineCarry(unsigned char a, unsigned char b);

/* This function makes function calls to determineCarry. Has no return value
 */
void runProblem4();

/* This function adds 2 signed chars and then determines if overflow occured.
 * This functions return TRUE if overflow occured and FALSE if overflow didn't
occur.
 */
static char determineOverFlow(char a, char b);

/* This function makes function calls to determineOverFlow. Has no return value
 */
void runProblem5();

/* This function adds 2 1 byte numbers, and determines if there's any carry. This
function returns the left most carry.
 */
unsigned int arbitrary_byte_add (unsigned char *result, unsigned char *a1,
unsigned char *a2, int size, unsigned int carry_in);

/*this function calls arbitrary_byte_add. It also allocates memory for the result
parameter used of arbitrary_byte_add. It has no return value.
 */
void runProblem6a();

/*this function calls arbitrary_byte_add. It also allocates memory for the result
parameter used of arbitrary_byte_add. It has no return value.
*/
void runProblem6b();
```

• Discussion of any difficulties encountered in the implementation, and information relative to issues of Reliability, Maintainability, and Security.

We had a hard time with how to use the makefile because we had multiple source files. We thought some of the directions were a little confusing, especially in problem number six, as we had to rewrite much of arbitrary_byte_add function due to this clarity issue.

• A summary of what you learned from the lab.

In this lab we learned about the sizes data (short, int, long) and how big/little endian format affects the resulting order of the data types. Next we learned about the difference between overflows and carries. Carry occurs when you don't have enough bits to represent the final number. Overflow occurs when you add two positive numbers together and you get a negative number, or vice versa.

```
Part 1: Data Types and their Sizes
========================
Byte Size: 1
Short Int Size: 2
Integer Size: 4
Long Integer Size: 8
========================
Part 2: Byte Ordering
========================
Byte Values of entire array (show as hexadecimal values, e.g., C7):
2a: 41 33 54 80 FF 99 01 78 55 20 FE EE 00 00 00 00
Short Int versions of the first four values of an array in hexadecimal:
2b: 3341 8054 99ff 7801
Int versions of the first two values of an array in
 hexadecimal:
2c: 80543341 780199ff
Long Int version of the first value of an array in hex:
2d: 780199ff80543341
2e: 0x0, 0x0, 0x0, 0x0, 0xee, 0xfe, 0x20, 0x55, 0x78, 0x1, 0x99, 0xff, 0x80, 0x54,
0x33, 0x41,
2f: 0x0, 0x0, 0xffffeefe, 0x2055
2g: 0x0, 0xeefe2055
2h: 0Xeefe2055
========================
Part 3: Unsigned Addition of Small Fields
========================
0x20 + 0x35 = 0x55
0x80 + 0x7F = 0xFF
```

0x80 + 0xFF = 0x7F
0xFF + 0x01 = 0x00
========================
Part 4: Detecting a Carry condition
========================
0x20 + 0x35 Carry: 0 (0 or 1)
0x80 + 0x7F Carry: 0
0x80 + 0xFF Carry: 1
0xFF + 0x01 Carry: 1
========================

Part 5: Signed Addition and Overflow
========================
0x20 + 0x35 Overflow: no overflow
0x80 + 0x7F Overflow: no overflow
0x80 + 0xFF Overflow: overflow
0xFF + 0x01 Overflow: no overflow
========================
Part 6: Performing Extended Field Arithmetic
========================
6a)
```
  0x44  0x00  0x00  0x00  0x01
+  0x30  0xFF  0xFF  0x00  0x00
=  0x74  0xFF  0xFF  0x00  0x01
```
Carry Out = 00
6b)
```
  0xFF  0xFF  0xFF  0xFF  0xFF
+ 0x00  0x00  0x00  0x00  0x01
= 0x00  0x00  0x00  0x00  0x00
```
Carry Out = 01
========================