CPSC 213, Winter 2010, Term 1 — Quiz 1 **Solution**

Date: Oct 6, 2010; Instructor: Tamara Munzner

1 For each of the following, give the smallest number greater than or equal to `0x1009` that is aligned as indicated.

1a Aligned for access to 2-byte shorts:

0x100a (decimal 4106) is aligned to 2-byte boundary because lower $log_2 2 = 1$ bit is 0. Equivalent statements: 4106 divided by 2 has no remainder, 4106 % 2 is 0.

1b Aligned for access to 4-byte longs:

0x100c (decimal 4108) is next number $\geq$ 0x1009 where lower $log_2 4 = 2$ bits are 0. Equivalent statements: 4108 divided by 4 has no remainder, 4106 % 4 is 0.

1c Aligned for access to 8-byte long longs:

0x1010 (decimal 4112) is next number $\geq$ 0x1009 where lower $log_2 8 = 3$ bits are 0. Equivalent statements: 4106 divided by 8 has no remainder, 4112 % 8 is 0.

1d Aligned for access to 16-byte chunks of memory:

0x1010 (decimal 4112) is next number $\geq$ 0x1009 where lower $log_2 16 = 4$ bits are 0. Equivalent statements: 4112 divided by 16 has no remainder, 4112 % 16 is 0.

2 What is does the following program output on a Little Endian (e.g., Intel) processor?

```
int main (char** argc, int argv) {
    char a[4];
    *((int*)(a)) = 0x01020304;
    printf ("a[2] = %d",a[2]);
}
```

The program outputs this string (you fill in the rest): `a[2] =`

The program outputs "`a[2] = 2`" because in Little Endian format the least significant byte of the number has the lowest address (i.e., comes first) and so a[0] is the least significant byte. The address of `a[2]` is the address of `a[0]` plus 2 and so `a[3]` is the third least significant byte (or the second most significant byte), which in this number is `0x02` or `2`.

3 Explain the tradeoff between static and dynamic arrays in C by giving one benefit of using each over the other.

3a **Benefit of static arrays** compared to dynamic arrays:

Accessing an element of a static array requires one fewer memory references than accessing an element of a dynamic array, because the compiler knows the base address of the array statically. And so, the only memory access required is to read or write the target array element. With a dynamic array, on the other hand, the compiler does not know the base address and so it must generate code to read this value from memory, thus requiring two memory accesses, one to read the base address and another to access the target element.

3b **Benefit of dynamic arrays** compared to static arrays:

Dynamic arrays are more flexible because their size can be determined at runtime. Since the size of static arrays must be determined statically, it can not depend on program input values or other dynamic properties of the program.

4 Give an RTL description and SM213 assembly code that a compiler might generate for this code. (There may be some code in between the declarations in the first line and the assignments in the last three lines, so `a` may have been assigned to any value.) Assign static values as the compiler would; explain values you use. The SM213 ISA documentation is provided on the last page for reference.

C:

```
int b[5], i, a*;

i = 2;
b[i] = 5;
a[i] = 3;
```

RTL:

$$
\begin{array}{lcl}
r[0] & \leftarrow & i \\
r[1] & \leftarrow & 2 \\
m[r[0]] & \leftarrow & r[1] \\
r[0] & \leftarrow & b \\
r[2] & \leftarrow & 5 \\
m[r[0]+r[1]*4] & \leftarrow & r[2] \\
r[0] & \leftarrow & a \\
r[0] & \leftarrow & m[r[0]] \\
r[2] & \leftarrow & 3 \\
m[r[0]+r[1]*4] & \leftarrow & r[2]
\end{array}
$$

SM213 Assembly:

```
ld $0x1018, r0 # load i loc into r0
ld $0x2, r1    # load 2 in r1
st r1, (r0)    # store 2 into i addr
ld $0x1000, r0 # load b loc into r0
ld $0x5, r2    # load 5 in r2
st r2, (r0, r1, 4) # store 5 in b[i]
ld $0x1014, r0 # load a into r0
ld r0, (r0)    # dereference a to r0
ld $0x3, r2    # load 3 to r2
st r2, (r0, r1, 4) # store 3 to a[i]
```

This code assumes compiler allocated the static variables in order starting at address `0x1000` *so that* `b` *is at* `0x1000`, `a` *is at* `0x1014` *and* `i` *is at* `0x1018`.