CPSC 213, Winter 2016, Term 1

Midterm II Sample Questions

Exercise 1

Implement the following C code in assembly. Pass arguments on the stack. Assume that r5 has already been initialized as the stack pointer and assume that some other procedure (not shown) calls `doit()`.
You do not have to show the allocation of x; just use the label x to refer to its address. Comment every line.

```
int x;

void doit () {
      x = addOne (5);
}
int addOne (int a) {
      return a + 1;
}
```

Exercise 2

Implement the following in SM213 assembly. You can use a register for c instead of a local variable. Comment every line.

```
int len;
int* a;
int countNotZero () {
      int c=0;
      while (len>0) {
            len=len-1;
            if (a[len]!=0)
                  c=c+1;
            }
            return c;
}
```

Exercise 3

Given the following code:

```
struct S {
  int*      a;
  int       b[4];
  struct T  c;
  struct S* d;
};
```

```
struct T {
      int  x[4];
      int* y;
    };
    struct S* s;
```

Write the SM213 assembly code that is equivalent to the following C statement

```
s->c.x[1] = 0;.
```

Exercise 4

What happens when the following code is compiled (and if it compiles) runs?

```
void gp (void* inv, void** outv) {
  intptr_t  in  = (intptr_t) inv;
  intptr_t* out = (intptr_t*) outv;
  *out = in * *out;
}

void fp (void* inv, void** outv) {
  intptr_t  in  = (intptr_t) inv;
  intptr_t* out = (intptr_t*) outv;
  *out = in + *out;
}

void foo (void** in, void**out, int n, void (*fp) (void*, void**)) {
  for (int i=0; i<n; i++) {
    fp (in[i], out);
  }
}

int main(int argc, char** argv) {
  intptr_t a[] = {2,3,4,5};
  intptr_t v   = 1;
  foo ((void**) a, (void**) &v, 4, gp);
  printf ("%ld\n", v);
}
```

Exercise 5

Consider the following code. Will it introduce a memory leak or a dangling pointer?

```
char* copy (char* from, int n) {
  char* to = malloc (n);
  for (int i=0; i<n; i++)
    to[i] = from[i];
  return to;
}
```

```
void foo (char* x, int n) {
  char* y = copy (x, n);
  printf ("%s", y);
}
```

Exercise 6

Write the interrupt driven version of the following code that prints the sum of the first 256 bytes of disk block 1234. Assume that async_read correctly performs the PIOs to request the block data be transferred into buf and that it enques sufficient information so that the interrupt service routine can call the specified event handler.

```
char buf[256];
void ps() {
  async_read (1234, buf, 256);
  int s=0;
  for (int i=0; i<256; i++)
    s += buf[i];
  printf ("%d\n", s);
}
```