Homework 4
CS 4481

Instructions You will turn in an electronic copy of this homework. The written answers should go in *hw4.pdf* and assembly code should go in *add.asm*, *fee.asm*, and *array.asm*. Zip all files into *hw4.zip* and submit using Moodle.

1. Memory layout affects the addresses assigned to variables. Assume that character variables have no alignment restriction, short integers must be aligned to halfword (2 byte) boundaries, integers must be aligned to word (4 byte boundaries, and long integers must be aligned to doubleword (8 byte) boundaries. Consider the following set of declarations:

   ```
   char a;
   long int b;
   int c;
   short int d;
   long int e;
   char f;
   ```

   Write a memory map (how the memory is laid out) for these variables with assumptions described below. For example, if `a` is laid out from bytes i to j and `b` is from k to l, then your answer would be "(i-j) a (k-l) b".

   (a) (5 points) Assuming that the compiler cannot reorder the variables.

   (b) (5 points) Assuming the compiler can reorder the variables to save space.

2. (1 point each) For each of the following types of variable, state where in memory the compiler might allocate the space for such a variable. Possible answers include registers, activation records, static data areas, and the runtime heap.

   (a) A variable local to a procedure

   (b) A global variable

   (c) A dynamically allocated global variable

   (d) A formal parameter

   (e) A compiler-generated temporary variable

3. (10 points) Write MIPS code that loads the values 3 and 4 into registers, adds them, and prints the result (the value 7). You will probably want to use the system call 1 (print_int). Use QtSpim to test it. Place your code in `add.asm`.

4. (10 points) Write MIPS code for the following C- code. You will need a label for the `fee()` function and will pass in formal arguments using the stack. Do not pass the parameters in with registers. Recall that the stack pointer grows *down*, that is, from large address to small address. Place your code in `fee.asm`.

   ```
   int fee(int a, int b) {
     return a+b;
   }
   void main() {
     print(fee(3, 4));
   }
   ```

   Your assembly code should have the following structure:

```
main:
        # put 3 and 4 on the stack using the register $sp

        # call fee. You'll probably want to use the jal instruction to
        # store the current address in $ra

        # load results from stack to registers

        # print result

        # exit

fee:
        # copy a and b from stack to local registers

        # add a and b

        # place result on stack

        # return using jr instruction
```

5. (15 points) Write MIPS code for the following C- code. You will not need a loop for allocating or initializing the array, but you will need a loop to sum the elements. Allocate the array on the stack and pass a dope vector as the formal argument to `fee()`. The dope vector will store the address of the array and the number of elements in the array. Note that it will store the address of the array and *not* the offset of the array from `$sp`. The dope vector will be on the stack. The multiplication instruction is `mul`. Place your code in `array.asm`.

```
int fee(int arr[]) {
  int sum = 0;
  for (int i = 0; i < length(arr); ++i) {
    sum = sum + arr[i];
  }
  return sum;
}
void main() {
  int a[3];
  a[0] = 1;
  a[1] = 3;
  a[2] = 5;
  print(fee(a));
}
```