McMaster University Comp Sci 4TB3/6TB3, Winter Term 2018/19 — Lab 11
For the Labs on March 27 - March 29,
Due Friday, April 6, 11 pm

Eden Burton, Jenny Wang, Spencer Park
out of 12 points

- *Submission is to be done exclusively through Avenue. Submissions via e-mail will **not** be accepted. A **10% penalty** will be accessed for each day the lab is submitted after the due date*.
- You need to download the P0 compiler posted together with **lab 9**.
- This assignment requires access to a Linux, MacOS X, or some other Unix computer. You can log in remotely to either `moore.mcmaster.ca` or to `mills.mcmaster.ca` with ssh. Submissions are tested on `moore.mcmaster.ca`, please check if your submission works there.
- In this lab, you are allowed to work in pairs, provided that you split the work equally and arrive at a common understanding of the solution. However, in that case you must state in your submission the person you worked with, such that similarities in the solution will not be construed as Academic Dishonesty. Working in groups of three or larger is not allowed and will be considered Academic Dishonesty. If you look for someone to work with, we will try to find a match, please contact the TAs.
- You are allowed and encouraged to talk to everyone in the course to get a common understanding of the problem, but you can share partial solutions only with your collaborator, if you work in a pair. The final submission must be your own, that is, you cannot submit identical submissions with two names on them.
- In the lab sessions, the solution to last week's lab questions are discussed and you can get help with this week's lab questions. Attendance at the labs is not checked.

The P0 language supports value and reference parameters. The P0 compiler passes all parameters on the stack. On the MIPS processor, by convention, registers $a0 to $a3 are used for the first four parameters, provided they fit in a word, and only any further parameters are passed on the stack. Consider:

```
compileString("""
        program parameterpassing;
          var a: boolean;
          var i, j: integer;
          procedure p(var u: boolean; v: boolean; w: integer;
                                var x, y: integer; z: integer);
            begin
              u := true; v := u and v; x := w + 3; w := 5; y := 7 - z
            end;
            begin p(a, true, 7, i, j, 9);
              if a then write(j)
            end
        """, 'P0.s')
```

Lab Question 1 (Annotating Generated Code, 8 points)**.** Modify the CGmips to use *a*0*to*a3 for the first four parameters. The code for above example should be:

```
        .data
j_:     .space 4
i_:     .space 4
a_:     .space 4
        .text
        .globl p
        .ent p
p:
        sw $fp, -12($sp)
        sw $ra, -16($sp)
        sub $fp, $sp, 8
        sub $sp, $fp, 8
        addi $t0, $0, 1
        sw $t0, 0($a0)
        lw $t7, 0($a0)
        beq $t7, $0, L1
L2:
        add $t4, $a1, $0
        beq $t4, $0, L3
L4:
        addi $a1, $0, 1
        b, L5
L3:
L1:
        addi $a1, $0, 0
L5:
        add $t2, $a2, 3
```

```
        sw $t2, 0($a3)
        addi $a2, $0, 5
        lw $t6, 4($fp)
        addi $t8, $0, 7
        lw $t1, 0($fp)
        sub $t8, $t8, $t1
        sw $t8, 0($t6)
        add $sp, $fp, 8
        lw $ra, -8($fp)
        lw $fp, -4($fp)
        jr $ra
        .globl main
        .ent main
main:
        la $a0, a_
        addi $a1, $0, 1
        addi $a2, $0, 7
        la $a3, i_
        la $t3, j_
        sw $t3, -4($sp)
        addi $t5, $0, 9
        sw $t5, -8($sp)
        jal, p
        lw $t0, a_
        beq $t0, $0, L6
L7:
        lw $a0, j_
        li $v0, 1
        syscall
L6:
        li $v0, 10
        syscall
        .end main
```

Test the generated MIPS code by running it in SPIM, http://spimsimulator.sourceforge.net/. Consider following procedures:

- genVar(x) takes the symbol table entry of variable x and returns a Var item with its location, which is either $fp relative for parameters and local variables or an absolute address for global variables. With parameter passing in registers, a variable can also be in one of $a0, ..., $a3. Function genVar(x) has already been modified accordingly.

- genAssign(x, y) generates code for x := y, where x is a Var item. Now x can be a Reg item: if y is a Cond, then 1 (for true) or 0 (for false) can be loaded directly in the register of x (rather than in a newly obtained register that is stored at the address of x); similarly, if y is a memory location, that can be loaded directly in the register of x; if y is a Reg as well, then y needs to be moved x.

- genFormalParams(sc): instead of assigning $fp and the offset on the stack to the reg and adr fields of all parameters, the first four parameters need to be assigned to registers $a0, ..., $a3.

- genActualPara(ap, fp, n): instead of "pushing" all parameters on the stack, for the first four parameters, registers $a0, ..., $a3 are used. There are three cases: the parameter a reference parameter and is already in a register, in which case the value needs to be moved to one of $a0, ..., $a3; the parameter is a reference parameter and needs to be loaded in a register, in which case one of $a0, ..., $a3 is used; or the parameter is a value parameter, in which case the parameter needs to be loaded in one of $a0, ..., $a3.