Homework #6: CMPT-379

Distributed on Tue, Nov 2; Due on Tue, Nov 9

Anoop Sarkar – anoop@cs.sfu.ca

(1) **Introduction to MIPS assembly**: The target of the code generation step for the compiler will be MIPS R2000 assembly language. MIPS is a reduced instruction set (RISC) machine. We will treat MIPS assembly code as a *virtual machine* and use a simulator for MIPS assembly called `spim` that takes MIPS assembly and simulates (runs) it on x86 Linux. `spim` is available in the location mentioned on the course web page.

Your task for this homework is to convert the following **`Decaf`** program called `catalan.decaf` *by hand* into MIPS assembly.

```
class Catalan {

    void main() {
       int i, j;
       callout("read_int", i);
       j = cat(i);
       callout("print_int", j);
       callout("print_str", "\n");
    }

    // catalan number of n
    int cat(int n) {
       int t;
       t = fact(n);
       return( fact(2*n) / (t * t * (n+1)) );
    }

    // factorial of n
    int fact(int n) {
       if (n == 1) { return(1); }
       else { return(n*fact(n-1)); }
    }
}
```

Provide the MIPS assembly program in a file called `catalan.mips` which should run on the simulator `spim` as follows: `spim -file catalan.mips`

When the MIPS program is run on the `spim` simulator, it should wait for an integer input *n* from the user, and then print out the result of the catalan function for *n* followed by a newline character. *The MIPS program must be a direct translation of the* ***`Decaf`*** *program.* Comment your MIPS code heavily. Compare your generated code to the parse tree and reflect on automating code generation (topic of a future homework). This exercise will familiarize you with MIPS assembly. Read the documentation provided on the course web page that introduces you to MIPS assembly, including a detailed tutorial on passing parameters on the stack frame for procedure calls in MIPS, and a tutorial on how to use `spim`. It assumes some familiarity with assembly language. Ask for background reading if you are not familiar with any of the terms used in the MIPS documentation.

You have to manage the register names used in the output assembly code. For this homework, you can ignore some of the complexities of code generation by assuming that we have a sufficient number of temporary registers at hand. Use the idea of using stacked temporary registers explained in Section 8.3 (page 480) of the Dragon book. A few facts that might be useful: in MIPS assembly, upto four arguments can be passed directly to a subroutine in the registers `$a0-$a3`, and `$s0-$s7` are temporary registers that retain values during a function call, while temporary registers `$t0-$t7` do not retain their values. The standard input-output library is provided through the `syscall` interface (compiled into `spim`).

I/O library service	`syscall` code	Arguments	Result
`print_int`	1	`$a0` = integer	
`print_string`	4	`$a0` = string	
`read_int`	5		integer in `$v0`
`read_string`	8	`$a0` = buffer, `$a1` = length	
`exit`	10		

Below is an example in MIPS that uses the `syscall` interface above to read an integer from standard input using `read_int`, and then prints it out to standard output using `print_int`, and then prints out a newline using `print_string`:

```
        .data
nl:
        .asciiz "\n"
        .text
main:
        li $v0, 5
        syscall
        move $a0, $v0
        li $v0, 1
        syscall
        li $v0, 4
        la $a0, nl
        syscall
```

I/O should be done only using the `syscall` service. Do not use the `jal printf` idiom used in some examples in the MIPS/`spim` documentation. Below is a simple **`Decaf`** program that computes a simple expression and the corresponding MIPS translation (it shows how to use temporary registers and how to store and use a global string constant).

```
class Expr {
  void main() {
    int x;
    x = 2*3+5;
    callout("print_int", x);
    callout("print_string", "\n");
  }
}
```

```
        .data
str0:
        .asciiz "\n"
#-------------------------
        .text
        .globl main
main:
        li $t0, 2
        li $t1, 3
        mul $t2, $t0, $t1
        li $t0, 5
        addu $t1, $t0, $t2
        move $a0, $t1
        li $v0, 1
        syscall
        li $v0, 4
        la $a0, str0
        syscall
```