LAB 9 Create MIPS code from your AST

Re-submit Assignment

Due May 8 by 11:59pm **Points** 300 **Submitting** a file upload **File Types** pdf

LAB -- Create MIPS code from your AST

Here is a nice additional resource http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Mips/stack.html (http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Mips/stack.html)

Click here for a file containing 3 MIPS and C code pairs

In this lab we will create MIPS code from our AST tree. The process is fairly straight forward. You should start from a copy of your AST printing routine and make changes to the copy to reflect the correct MIPS code. The goals of this lab include

*AT THIS POINT, YOU SHOULD NOT BE MAKING MANY CHANGES TO YOUR YACC FILE *

1) Creating MIPS code for IF/WHILE structures with jumps and labels
2) Incrementally creating MIPS code for the CMINUS language
3) Updating your Semantic actions to include a maximum size for each function (you will need this information to handle function call).
4) To learn about activation records, how to update registers to simulate recursion by managing stack pointers and storing the return address.

There are also some functionality actions required in this lab

A) update your main() program to use argv and argc so that it has the following actions
   1) if "-d" is given, then set the mydebug flag to 1, 0 is the default
   2) if given "-o", then the next argument is the name of the file you want to have MIPS dode created

```
    ./a.out -o foo
```

   will create a "foo.asm" output file.

   3) It is possible for your arguments to your compiler to have both options in either order

```
    ./a.out -d -o foo
    ./a.out -o foo -d
```

B) We will need to know the maximum offset for each function as we compile it. This will allow us to allocate enough run time stack for the function when we call it.
  --update your yacc code to identify the maximum amount of space needed and store this appropriately -- I stored mine in ASTNODE(FUNDEC)->value

C) Compiler tradition calls the output of assembly an emission, and uses "emit" as a prefix of function
  0) When we emit MIPS code, we have to emit 3 additional lines at the start of "main" The code is needed to update the global pointer so we know where to get global variables. The following MIPS code can suffice:

```
  main:
    #  we need to update the gp and sp so that we can call main with global variable --
    subu $sp, $sp, NUM  # number of bytes we need for the global data segment
    addi $gp, $sp, 0 # update the global pointer
    subu $sp, $sp, NUM1 # the number of bytes needed for main to run
```

  NUM is the global offset of all entities declared at the global level.
  NUM1 is the # of bytes main needs for its runtime stack

  3) Copy your ASTprint to EMITAST, add a paramater for the file pointer, get rid of level
  4) Update EMITAST to output MIPS code based on the type of AST node you are encountering.

 Start with write, and read.

Test with
  write 5
  write 5 + 3

  read x

D) Functions require 4 separate steps, called the "subroutine call linkage."   The sequence is:

a) Calling routine saves the current stack pointer
   Copies the function arguments to memory where the function will expect them
   Advances the stack pointer
   executes the JAL MIPS instruction

b) The Function entrance stores the return $ra in memory, I put mine in ($sp)

c) The function on return stores the return value in $v0
   restores the $ra value from ($sp)
   retores the stack pointer from 4($sp)
   Jumps to $ra

   For all functions, you need to emit a function return with $v0 at the end of the function defintition.

d)   The returned value in $v0 is copied to the appropriate memory location if needed
  -- you will need to write subroutines for each of these steps

Expression notes

Mult $t0 $t1   -- stores the result in HI and LO registers. We will not look at overflow, so we need to just store LO like
   MULT $T0 $T1
   MFLO $T0 -- $TO now has the lower bytes of the result of the multiply

DIV -- look at the opcode sheets to pull the value for / and %

Since this assignment can be very difficult, you can choose the level of grading to receive.

DELIVERABLES

All programs must be able to read a variable and write a variable or number. Programs which cannot do this action, will lose AT LEAST 60%

a) Can only read and write variables and numbers 40% off
b) A program which calls no functions and does no array arithmetic at least 30% off
c) Can do a-b and "if" and "while" 20% off
d) Can do a-c and also arrays   10%
e) Can do a-d and handle function calls
f) Add strings to "write", Extra Credit 5%... You MUST do a-e do get the extra credit

You need to include your EMIT code, a copy of the test code you used, a copy for the MIPS output and screen shot of the CONSOLE from the MIPS simulator

If you are in category d-f, you must ALSO show me or the TA your code in operation. For other categories, we may ask you to show us your code in operation if we cannot determine that your code generated the MIPS code

LATE SUBMISSIONS WILL FOLLOW THE LATE POLICY