CSI 333 – Programming at the Hardware-Software Interface – Fall 2017  
Project IV

The total grade for the assignment is 100 points.

You must follow the programming and documentation guidelines available in the Blackboard module Projects.

This is a **team** project (except for those students who have opted to work on their own). Group of **two** students may work on this project together.

# Description:

You are required to write a C program

* whose input is a MIPS Assembly Language (MAL) program and
* whose output is a list of labels of the MAL program used as variables, a list of labels of the MAL program used for the flow control, or both.

Details regarding MAL programs and lists of labels are given below.

Command Line Details: Suppose the executable version of your program is named prog4.out. The program will be executed by a command line of the following form:

prog4.out flag inputfile outputfile

In the command line above, the arguments inputfile and outputfile specify the names of the input file (which contains a MAL program) and the output file respectively. A valid flag argument is one of -v, -f or -b.

* If the flag is -v, your program must produce only a list of labels of the MAL source program used as variables in the specified output file. (Variable labels are identifiers defined in the .DATA segment of a MAL program.)
* If the flag is -f, your program must produce only a list of labels of the MAL program used for the flow control in the specified output file. (Flow control labels are identifiers defined in the .TEXT segment of a MAL program.)
* If the flag is -b, your program must produce (i) a list of labels of the MAL source program used as variables, and (ii) a list of labels of the MAL program used for the flow control in the specified output file.

Details Regarding MAL Programs:

1. Each line of a MAL program may be
   1. a blank line (containing only white-space characters),
   2. a comment line (a line that starts with the character ’#’) or
   3. a MAL statement which has the following fields separated by one or more spaces or tabs:
      1. an optional label field terminated by ’:’,
      2. a mandatory opcode field,
      3. an optional operand field consisting of zero or more operands separated by commas, and
      4. an optional comment field starting with ’#’.
2. Every line of a MAL program is terminated by the newline (’\n’) character and has at most 80 characters (including the newline character).
3. Any line that has ’#’ as the first character is a comment line.
4. The labels start from the first position, thus, a line that starts with a space or tab does not have a label.) The maximum number of labels that can appear in any MAL program is 100.
5. The first character of any identifier is a lower- or upper-case letter or the underscore character followed by zero or more upper/lower case letters, digits or underscore characters. The identifiers are case sensitive. (Thus, Val and val represent different identifiers.) The maximum length of an identifier is 10.
6. Opcodes are not identifiers. Also, operands starting with the symbol ‘$’ are not identifiers; they represent registers of the MIPS machine.
7. If the operand field of a MAL program starts with the single quote character (’) or the double quote character ("), then the operand field does not have any identifiers. (The reason is that an operand field starting with the single quote specifies a literal character; an operand field starting with the double quote character specifies a string.)
8. An identifier in a MAL program:
   1. is defined in the source line where the identifier appears as a label and
   2. is used in lines where it appears in the operand field.

Examples:

iloop\_beg: la $7,arr #Get address of arr into register 7.

In the above, the label field has the identifier iloop\_beg. The opcode field has la. The two operands are $7 (a register – not an identifier) and arr (an identifier). The operand field is followed by the comment field that starts with the # character. The above statement defines the identifier iloop\_beg and uses the identifier arr.

loop: sw $15,avg #Store reg. 15

In the above instruction, the label is loop, the opcode is sw and the operands are $15 and avg. The instruction defines the identifier loop and uses the identifier avg.

An example of a MAL program and lists of labels are shown below. Be sure to study the example given below before writing your program.

Suppose the file example.mal contains the following MAL program.

#A sample MAL program.

.data #Data segment begins here.

avg: .word #Will store the average.

i1: .word 20 #First integer.

i2: .word 13 #Second integer.

i3: .word 82 #Third integer.

prompt: .asciiz "Value is: "

nl: .byte ’\n’

.text #Text segment begins here.

\_\_start: lw $15,i1 #$15 contains 20.

lw $16,i2 #$16 contains 13.

i10: add $15,$15,$16 #Operand field has no ids.

lw $16,i3 #$16 contains 82.

add $15,$15,$16 #$15 contains the sum (115).

li $16,3 #$16 contains 3.

div $15,$15,$16 #$15 contains the average (38).

i20: sw $15,avg #Store the average.

puts prompt

put avg

putc nl

sw $15,avg

la $16,i1

sw $15,0($16)

add i3,i3,1

done #Similar to halt.

After the execution of the program using the command line

prog4 -b example.mal output.lst

the contents of the output file output.lst should be as shown below. In studying the output file, you should keep the following in mind:

1. Name of the identifier in the list is surrounded with ‘-‘.
2. The identifiers in the list are in order of their definition in the program.
3. For each identifier, the source lines where it is used are in order of their appearance in the program.
4. If an identifier is not used, there is neither source line, no empty line after the identifier.
5. When an identifier is used several times in a source line (e.g. the identifier i3 appears twice in one line), the source line appears only once in the list.

Contents of the file output.lst:

Variable ID –avg-

i20: sw $15,avg #Store the average.

put avg

sw $15,avg

Variable ID -i1-

\_\_start: lw $15,i1 #$15 contains 20.

la $16,i1

Variable ID -i2-

lw $16,i2 #$16 contains 13.

Variable ID -i3-

lw $16,i3 #$16 contains 82. 6 add i3,i3,1

Variable ID –prompt-

puts prompt

Variable ID –nl-

putc nl

Flow Control ID -\_\_start-

Flow Control ID -i10-

Flow Control ID -i20-

Programming Suggestions:

1. Data structure to be used: array (of size 100) of struct, where each struct has the following data members:
   1. a char array of size 11 (to store an identifier),
   2. a pointer to a linked list; each node of the list stores a source line where the identifier is used.
2. Don’t assume any limit on the number of lines in the input file. Read the input line by line (using fgets). For each line of the input file which is not a comment line or a blank line, use strtok to parse the line (i.e., extract the various fields of the instruction).
3. All input files will satisfy the following conditions.
   1. The MAL program in the input file won’t contain any errors.
   2. Every identifier will be defined somewhere in the MAL program. (However, it is possible that an identifier is not used anywhere.)
   3. There will be no multiply defined identifiers.
4. You need to check only for the usual command line errors (wrong number of parameters on the command line, invalid flag, the input or the output file can’t be opened). In such a case, your program must output a suitable error message to stderr and stop.
5. The input and output file names must NOT be hard-coded into your program. These files names are available to your program as Unix command line arguments.
6. Program must contain several functions in addition to main.
7. It is advisable for each team member to take the responsibility for one of the program’s major components:
   1. Processing each MAL statement (e.g. checking if the statement is a comment, a blank line or a MAL statement, identifying the label and/or the identifiers appearing in a MAL statement).
   2. Maintaining the data structure that stores the lists information.

# Electronic Submission:

There are two mandatory submissions for each project. Both submissions are the same files but to be used for different purposes. You may do both submissions at one go:

* source code for the evaluation using turnin-csi333 on the ITS Unix machine,
* source code for the record – the standard Blackboard procedure for the assignments.

For the team project, each team must make only one turnin-csi333 submission. That is, in each team, ONLY ONE member must do this. Team submissions must include additional documentation in the source file as explained below.

But the Blackboard submission must be done individually by EACH STUDENT even for the team project.

Important Notes: ignoring any of the following rules will result in penalty or even ZERO grade for the project.

* 1. For Project 4 you must turn in the file **named** “**p4.c”**.
  2. At the top of your C source file the following information must appear in the form of comments:
     1. For students working by themselves: You must have the following information at the beginning of your source file (p4.c) in the form of comments: your name, your Unix login id and the day/time of your weekly discussion class.
     2. For students working in a team: You must have the following information at the beginning of your source file (p4.c) in the form of comments:
        1. The names and Unix login IDs of the two team members and the day/time of the weekly discussion class.
        2. A clear explanation of how the work for the project was divided among the two team members. Indicate clearly who developed each function and how the testing work was divided between the team members.
  3. Make sure that your programs compile and produce correct results on the Unix machines (itsunix.albany.edu) supported by Information Technology Services (ITS) unit of UAlbany. Programs that cause compiler or linker errors on the ITS Unix machines will NOT receive any credit.
  4. Using the turnin-csi333 program as discussed below is the ONLY acceptable way of submitting programming assignments in this course. You should NOT email the files to the instructor or to the TAs.
  5. Remember that you must submit only your C source files. DON'T turn in unnecessary files (e.g. object files with extension “.o” created by compiling C source files, executable files such as “a.out”, etc.).

To submit your files electronically, you must have the source files on one of the ITS Unix machines. For this project, the file p4.c must be in your working directory and you must be logged on to one of those machines to carry out the electronic submission.

To perform submission, you should type the following command to the Unix operating system:

turnin-csi333 -c csi333 -p hw4 p4.c

After you issue the above command, the system responds with:

The sections of csi333 are:

WE\_0920

WE\_0415

FR\_1130

FR\_0140

MO\_1025

Enter your section:

Depending upon the day and time of your weekly discussion section, you would type the appropriate section. For example, if your discussion class meets on Wednesdays at 09:20 AM, you would type WE\_0920 followed by the return key. The system will then respond with

Your files have been submitted to csi333, hw4 for grading.

In the above message, "hw4" refers to the name of the project that is currently active.

If you submit your program during two days after due date the name of the project should be   
"hw4-late":

turnin-csi333 -c csi333 -p hw4-late p4.c

Attempts to submit the program or check submission after the two-day grace period will fail. Lateness penalty is 10 points per day.

Additional information about the turnin program:

* 1. If you use the turnin command above again later (before the deadline), then the files submitted previously would be completely replaced by the newly submitted files.
  2. You can obtain the names of the files that you have submitted to the current project using the following command:

turnin-csi333 -c csi333 –v

* 1. You cannot submit your work if the project is closed. The project is starting from the date the homework is on Bb till the due date. To see the list of open/closed projects, use the following command:  
     turnin-csi333 -c csi333 -l

# Program Grading:

Programs will be graded using a script written by the TAs. The script will compile your source program, generate the executable version and run the executable on new test data. The TAs will grade the version that you submit; once the submission is closed, you won’t be allowed to make any changes to your program.

For students working individually:

1. Correctness: 85 points
2. Structure/documentation: 15 points

For students working in a team:

1. Correctness: 65 points
2. Structure/documentation: 15 points
3. Team work: 20 points

Each team member must participate in developing, documenting and testing the program. Each team should include additional documentation at the beginning of the source file indicating how the work for the project was divided between the two team members. (Indicate clearly who developed each function and how the testing work was divided between the team members.) After the submission deadline, each team must meet with their TA. During the meeting, the TA will ask questions about the team’s program and determine the points for team work. (The two team members may receive different scores for team work.)

# Some sample data to test your program:

Important Note: Some sample inputs that can be used to test your programs are given below. However, you should remember that when we compile and run your source files, we will use other data. Just because your programs work for the sample inputs given below, you shouldn't assume that they will work for all inputs. Therefore, you should test your programs thoroughly with other input values.

The content of input file mal\_prog.txt:

.data

x1: .word 0

x2: .word 80

.text

begin: lw $15,x1

add $15,$15,x2

beqz $15, end

sw $15,x1

end: done

Suppose we execute the following Unix command:

prog4.out -b mal\_prog.txt output.txt

The contents of the file output.txt:

Variable ID -x1-

begin: lw $15,x1

sw $15,x1

Variable ID –x2-

add $15,$15,x2

Flow Control ID –begin-

Flow Control ID –end-

beqz $15, end

In addition, the test file test1.mal and the corresponding output file test1\_output.txt will be provided to help you to test your project. **It is not the test file we will use to grade your project!**

The file test1.mal represents a sample input file containing a MAL program. The file test1\_output.txt is the result of executing the following Unix command where "prog4.out" is the name of the executable version of your program):

prog4.out -b test1.mal test1\_output.txt

Since the above command line uses the flag "-b", that the file test1\_output.txt contains both the lists. If we use the flag "-v" on the Unix command line above, the output file must contain ONLY the list of variable IDs; likewise, if we use the flag "-f" on the Unix command line above, the output file must contain ONLY the list of flow control IDs.