Final Project

- a C programming component
- an assembly language programming component
- a formal written project report
- an informal 5-minute in-class presentation, illustrated with sample code (to be given during the last two class sessions)
- [only for team projects] a report summarizing, in detail, the contributions of each team member

Team projects require only a single project report and a single team presentation.

Program Format

All programs (both C and assembly language) must be fully and correctly commented (“correctly” means you have deleted any of my comments that say things like “*** INSERT CODE HERE ***” and that comments are accurate). All code adapted from outside sources must be cited (author and source) in the program comments as well as in the final written report. If several team members contribute to the code, each member’s contribution should be briefly explained in the comments as well as in the team report (with more detail given in the team report). Every program should cite the honor code statement from the syllabus.

Code should “look nice” — correct indenting, white space to separate logical divisions in the code, readable code, etc. Please do not use tabs in your C program if it means your code will be bunched up against the right margin or if it means many lines will wrap around; if you like to use tabs, do a global search and replace to substitute each tab with some small number of spaces (2 or 3) before handing in your program.
Report Format

The project report (and team report if applicable) must be in PDF format; Don’t submit .odt or .txt files. The project should have a title (e.g., “Translating Multiplication and Division into NASM Code” and should list the author(s). It should be organized into sections (e.g., “Project Description,” “Explanation of Code,” “How to Run the Project,” “Lessons Learned,” “References”). The report should include snippets of code to use as examples—these should be set off from the text in displays with fixed-width font (such as Courier) and labeled. For instance,

**Explanation of Code.**

... The third program, `proj3.asm`, illustrates how to generate the MIPS code for simple arithmetic expressions. For example, Figure 1 shows the code for processing an input string consisting of numbers and operators, separated by spaces. Variable “`word`” holds the next object (integer or operator) and is obtained through repeatedly calling the “`strsep`” function.

```c
while ((word = strsep(&linepointer," "))!= NULL) {
    if (strlen(word) == 0) continue;

    if (! isdigit(word[0])) {
        op = word[0];
        ... etc. ...
    }
}
```

![Figure 1: Processing simple arithmetic expressions](image)

**The Project Itself**

The various options were described on the lab 8 handout.

The remaining labs this semester will include project time along with abbreviated lab assignments.

You are not expected to figure out everything by yourself—I expect (indeed, require) that you consult with me, ask questions, and generally seek my help and advice throughout the entire process. You must provide two one-page informal progress reports, one due on Monday, Nov. 23, and a second due on Thursday, Dec. 3. These should be specific about what you have completed so far, what obstacles you have encountered, what questions you have, etc.

**Final Comments**

This is not supposed to be an enormous project in terms of the amount of code you need to write. However, it should represent a significant learning experience. You should demonstrate that you know how to do “new things” (such as calling NASM functions from a C program or
correctly processing complicated arithmetic expressions involving variables, constants, operators, parentheses, etc.).

Start easy! Don’t begin by trying to do everything. For instance, if you have chosen option 1 (from lab 8), start with simple integer expressions involving just addition and subtraction; generate simple MIPS code just involving a couple of registers. From there, try adding assignment statements and generate “.word” or “.space” commands and “lw” and “sw” commands that refer to them. Then try adding parentheses (this may be too hard—in that case, try something easier, like adding logical “or” or “and” operators).

Never, ever allow yourself to become so intimidated by the project that you stop working on it! Instead, AS SOON AS you feel out of your depth, make an appointment to talk with me!