In this project, you will be writing assembly language programs, using the Y86 assembly language detailed in your textbook (with slight modifications). Your assignment is to take the C programs we give you and write equivalent assembly programs for each one. You should also aim to have your programs run in as few instructions as possible, so you may need to perform some optimizations to the programs either before or after translating them from C.

1 Procedure

1.1 Obtain the project files

For this project, we have supplied both a tarfile with files you will be working with, and a Y86 assembler and simulator. Both the yas and yis programs discussed in lecture are available in the ~/216/public/bin directory, so they should already be in your path. The tarfile contains a .submit file and three C programs (prog1.c, prog2.c, and prog3.c) that you must translate to Y86 assembly programs. Extract the files from the tarball into your ~/216/project4 directory, using a procedure similar to the one you used for prior projects.

1.2 Create the assembly source files

For each C program, you must create a Y86 assembly language program that functions in a similar manner (meaning it produces the same output as the C program if both are given identical input). Your primary goal should be to produce a working version of each program. As a secondary goal, you should also attempt to have your assembly programs execute on the simulator with as few steps as possible, as measured by the simulator’s output. A small portion of your grade will depend on the efficiency of your programs.

Your Y86 source files should be named the same as their corresponding C source file, but with a .ys extension; for example, prog1.ys is the Y86 assembly code for prog1.c.

2 Specifications

The three C programs you will translate are: (a) prog1.c, a program that determines the greatest common divisor of two positive numbers; (b) prog2.c, a program that reads in n numbers and a key, and partitions the numbers into three sets based on the key; and (c) prog3.c, a program to calculate the nth Fibonacci number, recursively.

All your programs are expected to terminate via a halt instruction; abnormal termination should never occur (except in the case of an I/O error, which you are not required to handle). Also note that each program prints a newline at the end of execution; this is to guarantee that the output is distinguishable from the simulator’s normal output. As such, it is imperative that you do not leave it out.

You may, in all cases, assume that legitimate integer numbers are read in. However, you must make sure your programs work for the ranges of values specified for each program; please pay attention to those ranges.
2.1 Greatest common divisor program
This program reads in two positive integers from standard input, and then prints out their greatest
common divisor, followed by a newline.
You may assume that the value of the numbers are in the interval $[1, 1000]$.

2.2 Number list program
This program begins execution by reading two integers from standard input, $n$ and $key$. The program
then reads in $n$ additional integers from standard input; once $n$ integers are read, each of the $n$
integers is printed out (separated by a space) in order based on the key. More specifically, first all the
integers with values less than that of the key are printed, then those with values equal to the key, and
last all the those with values greater than that of the key. In addition, each set of numbers should be
printed in the same order as they were in the input. Finally, the program prints a newline.
You may assume that the value of $n$ is in the interval $[0, 500]$. Also, your program should not have
a space at the beginning or end of the line (a fencepost problem); note that the C code you are given
does not handle this requirement correctly. You will lost some points if you have such extra spaces.

2.3 Fibonacci program
This program begins execution by reading a single integer (called $n$) from standard input. It then
prints out the $n$th Fibonacci number ($F_n$), followed by a newline, via a recursive algorithm. For the
purposes of this assignment, we define $F_0 = 0$, $F_1 = 1$; for all other values of $n$, $F_n = F_{n-1} + F_{n-2}$.
You may assume that the value $n$ is in the interval $[0, 14]$ (so that it doesn’t take too long to simulate).
Note that you must implement this program in a recursive manner, and not iteratively.

3 Important Points and Hints
1. The $yis$ simulator has a limit of 100000 steps per execution. While your programs should not
take anywhere near that many steps to run, keep it in mind when designing your programs, so
that they all stay within this boundary for all valid inputs.

4 Grading Criteria
Your project grade will be determined by the following:

<table>
<thead>
<tr>
<th>Component</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Results of public tests</td>
<td>25%</td>
</tr>
<tr>
<td>Results of secret tests</td>
<td>55%</td>
</tr>
<tr>
<td>Code style grading</td>
<td>15%</td>
</tr>
<tr>
<td>Code efficiency</td>
<td>5%</td>
</tr>
<tr>
<td>Extra credit</td>
<td>5%</td>
</tr>
</tbody>
</table>

Instructions on how to run the public tests will be provided along with the tests, in a file named
README (included in the tarfile). As before, public tests will be released shortly after the project’s
release, and secret tests some time after the late deadline.

4.1 Style grading
For this project, some style guidelines are obviously different, as you are writing in assembly lan-
guage, not C. Please pay close attention to these guidelines:

1. All Y86 source code files should begin with a comment containing your name, University ID
number, and Grace login/Directory ID.

2. Lines should still be no longer than 80 columns.

3. Reasonable and consistent indentation is still required, although the indent program will likely not be of much use to you. As these programs should be relatively short, it should not be too difficult to manually maintain indentation.

4. Label names should be descriptive and meaningful.

5. Your code must be thoroughly commented. Assembly language can easily become unreadable without proper documentation, so it is absolutely necessary that you comment your code.

6. Use appropriate whitespace, especially between blocks of instructions that are performing different tasks.

4.2 Code efficiency

Running the yis simulator on your assembled code will print the number of steps the simulator takes to execute your program. You are expected to, once your code is working, attempt to revise it to lower this count, so that your code is more efficient. You may also attempt to optimize the C code and then translate it; this may yield better results.

Since the programs have ranges of valid inputs, we will usually be more concerned with your code’s efficiency toward the high ends of those ranges rather than at the lower ends.

4.3 Extra credit

Once again, we offer extra credit to students who make an early submission that passes all public tests. You will receive 5 points of extra credit if you make a submission by 6PM on Tuesday, November 2, that passes all public tests for the project.

5 Submission

5.1 Deliverables

The only files we will grade are your three Y86 source files (prog1.ys, prog2.ys, and prog3.ys). Any changes made to the C files will not affect your submission or grade at all.

5.2 Procedure

As for previous projects, executing “submit” in your project directory will submit your project. We once again encourage you to run public tests on Grace rather than submitting to the submit server and waiting for your submission to be evaluated; it is much faster for you to see your results if you run the tests yourself, and the submit server works much more quickly if the class makes fewer submissions.

6 Other Notes

6.1 Academic Integrity

As described in the syllabus, any evidence of cheating will be referred to the Student Honor Council and may result in a grade of XF in this course. Submissions will be checked with an automated source code comparison tool to look for evidence of cheating.
6.2 Deadlines
Submission deadlines are strictly enforced by the submit server, and we will not extend them for things such as network outages. Extensions may be given on a case-by-case basis, but will likely only be granted in emergency cases. Therefore, you should start work on this project soon, as last-minute technical problems are not an excuse for missing either the on-time or late deadline. We would also like to remind you that this project is not trivial, so we recommend you begin it as soon as possible.