**Processor Project**

**1 General Description**

As a team, you will design a "miniscule instruction set" general purpose processor that can execute programs stored in an external memory. You will also model your design, test it, debug it, and assess its performance. You may choose to implement it on a Field Programmable Gate Array (FPGA) microchip. Throughout the project, you will maintain current documentation for your design, as well as an ongoing record of your design process. Each team will present their design to the class.

The project is organized into regularly scheduled milestones that will guide you through a top-down design process. For each milestone, you will submit your current design documentation and an activity journal.

This is a big project and the milestone schedule allows little room for error. Later milestones tend to require more effort. If at all possible, you should work ahead of the schedule and even complete the project early.

**2 Learning Objectives**

In the process of completing the final project, students will develop their abilities to:

* Apply the principle of abstraction in analysis and design problems.
* Specify, design, test, and document instruction set architectures and their hardware implementations, taking into consideration key computer organization design principles
* Work effectively as members of a team.

**3 Completion Requirement**

Because of the central role that the project plays in satisfying the learning objectives of this course, students must satisfactorily complete the project in order to pass the course.

**4 Teams**

The project will be completed in teams of 3 or 4. Teams will be formed using your input. Here are factors that may be considered when forming teams:

* Who you don't want to work with
* Who you do want to work with
* Making sure small groups can succeed
* Your skill level with the material

**5 Weekly Meetings**

When the project begins, each team will schedule a weekly design review meeting with their instructor. Plan for meetings to last about 15-30 minutes. Your instructor may use the design review meetings to quiz you about your design, so be prepared.

**6 Requirements**

Your processor must be capable of executing programs stored in an external memory with which it communicates using:

* A 16-bit address bus,
* A 16-bit data bus,
* A mechanism for basic input (i.e. you should not need to recompile your design to change the input to your demo code), and
* A mechanism for basic output (i.e. it should be easy to see the results of your computation).

Your instruction set:

* Must be capable of performing general computations, and
* Must support parameterized and nested procedures.

Additionally, your processor should support some of these:

* Implementation on an FPGA board,
* Interrupts from two input devices,
* Reading from an input port which is at least 4-bits, and
* Reading from and writing to a special 16-bit display register, and displaying the contents of the display register on the LCD display via a 16-bit output port.

Note that this means your instructions should not exceed 16bits in length. If you feel you need larger instructions, consult your instructor. All other design choices (RISC vs CISC, number of instructions, etc.) are yours to make. Designs will be evaluated based on their ability to perform general-purpose computations, the performance on those computations, the resources required and the interestingness of the design. See the project rubric for details on how projects will be evaluated. The rubric is not intended to be exhaustive.

In order to demonstrate that your processor has met the requirements, you must implement a program that computes the relatively prime value for a number using the Euclid algorithm.

**7 Interrupts and I/O**

If you choose to implement interrupts and generalized I/O, the details below should guild you. If interrupts are supported, the main program would consist of initialization followed by an infinite loop, and the program would respond to interrupts from two input devices. The interrupt mechanism facilitates the communication between the processor and the outside world.

* When an interrupt from the first device occurs:
  + Read the input port,
  + Replace the contents of the display register. If the input port is less than 16-bits, the input port value will need to be combined with the previous contents of the display register so that a 16-bit number can be built up, and
  + Write the contents of the display register to the output port.
* When an interrupt from the second device occurs:
  + If the display register is zero,
    - Do nothing.
  + Otherwise,
    - Read the display register,
    - Call the procedure relPrime with the contents of the display register as an argument,
    - Write the result back into the display register, and
    - Write the contents of the display register to the output port.

relPrime uses Euclid's (as a procedure) to find a positive integer that is relatively prime to the argument.

**8 Euclid's algorithm**

For two numbers to be relatively prime, their greatest common divisor (gcd) must be one (i.e. they must have no common divisors other than one). Euclid's algorithm is used to determine the gcd of two numbers. The following is a high-level language specification for a program that your processor must be capable of executing. The program determines the relatively prime value to an input number using Euclid's algorithm.

// Find m that is relatively prime to n.

int

relPrime(int n)

{

int m;

m = 2;

while (gcd(n, m) != 1) { // n is the input from the outside world

m = m + 1;

}

return m;

}

% http://dystopiancode.blogspot.com/2012/08/euclids-gcd-algorithm-in-ansi-c.html

// The following method determines the Greatest Common Divisor of a and b

// using Euclid's algorithm.

int

gcd(int a, int b)

{

if (a == 0) {

return b;

}

while (b != 0) {

if (a > b) {

a = a - b;

} else {

b = b - a;

}

}

return a;

}

Note: There are many ways to determine a relatively prime value to a given number. This particular specification is chosen to test the performance of your processor.

**9 Documentation**

As you design your processor, keep the following in mind:

* Your project will be evaluated on the basis of how well it demonstrates that you have satisfied the learning objectives of the course. The operative word is demonstrates, and the mechanism for that demonstration is your documentation. Be thorough, clear, and concise.
* Your project will also be evaluated on the basis of its ability to perform general-purpose computations, the performance on those computations, the resources required and the interestingness of the design. Following the design principles presented by the Patterson and Hennessy textbook will help you make good tradeoffs.

Documentation is an important part of your project. This is how you will communicate your ideas to the rest of world. Therefore, it is important to document your design process as well as your design. For this project, you will do it through 3 documents.

**Design Document:**

Describes the design in sufficient detail for an outsider to completely understand it. More specific suggestions along these lines are included in the Milestones section below. This part of the documentation should be continuously updated to reflect the current status of the design.

**Design Process Journal:**

Records the process by which the current design was developed. An outsider should be able to read this document and know the design options that have been considered, the design decisions that have been made, and the rationale for those decisions. This part of the documentation should be continuously appended to and reflect the complete history of the design.

All documentation should be high quality product in electronic form. ASCII, text and handwritten documents are not acceptable. PDF, html and Word are examples of acceptable formats. Neat and legible, scanned diagrams are acceptable for all but the final project presentation and final project report.

All materials will be submitted to your team's repository. Make sure that your design documents are easily recognized. **Never check in multiple design docs.** You may be penalized if your instructor cannot find your design. Keep your repository organized. For example, after completing Milestone 5, your directory may look like this:

Design <dir>

Design document.pdf

Design journal.pdf

Implementation <dir>

... many processor files ...

**10 Milestones**

The project is organized into regularly scheduled milestones that will guide you through a top-down design process. For each milestone, you will specify your processor at the next lower level of abstraction, verify that your new specification satisfies the requirements imposed by the previous level, and state the requirements for the next level of specification. For example, the first part of milestone one essentially requires you to

* Write an assembly language programmer’s manual for your processor, and
* Use your assembly language to write the program described in the Requirements section above.

By doing so, you are

* Specifying your processor at the assembly language level of abstraction, and
* Verifying that your assembly language specification meets the project requirements.

This is a big project and the milestone schedule allows little room for error. Later milestones tend to require more effort. If at all possible, you should work ahead of the schedule and even complete the project early.

**11 Presentations**

Each team will present their project to the class. Presentations should include an overview of your design, highlighting unique or interesting features and a demonstration of its functionality (i.e. run it for us). Each team member must contribute in a substantial way. Presentations will be peer reviewed.

**12 Final Documentation**

Your final report will be a substantial document. It should include a cover page, a table of contents, an executive summary, an introduction, the body, a conclusion, and appendices. The body should summarize and discuss your instruction set design, your implementation, your Xilinx model, your testing methodology, and your final results. The appendices should include your complete design documentation, design process journal, and test results. Except for the appendices, the report should be written as if the reader is familiar with computer architecture, but not with this project.