The HMMM Interpreter
Project Plan

Hasana Parker, Alex Martin, Henry Merrilees

October 23rd, 2023

1 Preface

Problem Statement Consider a CS5'er immediately following the HMMM unit. At this point, they would know how a human might go about executing a list of assembly instructions. Still, they would have yet to learn how computers actually translate and execute assembly. So how might they acquire a substantive mental model?

Proposed solution Conventionally (as in E85), you first learn a hardware description language, then implement a bare-bones microprocessor. (With the assembler left as exercise for the reader.) Unfortunately, learning a hardware description language is unintuitive and beyond the scope of CS5 or similar courses. Fortunately, CS5'ers are capable of reading algorithms from already-written python. So we can utilize python to "describe" a basic microprocessor, component by component, using conventional programming idioms in place of non-trivial hardware equivalents. Understanding how assembly instructions are converted to machine code and executed is critical to demystifying computers to the developing computer scientist. Even if one has no interest in electrical engineering, having the rudimentary knowledge/awareness of the functioning of a microprocessor is also essential to learning programming topics such as caching and branch prediction for the purposes of performance optimization.

2 Introduction

Nominally, our tutorial is to teach the user how to write an emulator using python. Practically, we use python as an abstract vehicle to explain at a high level how a microprocessor works. We also implement a basic web interface both for usability and for student edification. If done well, the our tutorial may be "hollowed out," turned into an assignment (i.e., supplanting code with well-motivating function comments sufficient for the code to be re-created by the student alone).

3 Tutorial Outline

Our project thus features two major aspects - developing a simple HMMM interpreter in Jupyter, and then using Voila and IPywidgets to turn the interpreter into a web app. Building the emulator will require no tools beyond Jupyter, as the goal of this project is to describe the emulator with python. The web app component will likewise be designed to be as simple as possible, requiring no prior knowledge of web apps - rather it will be a simple way of publishing the completed interpreter such that someone can interact with it. As such it will use Voila and ipywidgets, which allows for the direct conversion of Jupyter code blocks into web-based interactables. Specifically, ipywidgets is a library that allows for the creation of different interactive items such as sliders, buttons, and tabs from code cells. Voila then takes these widgets and compiles them into a succinct and effective web app.

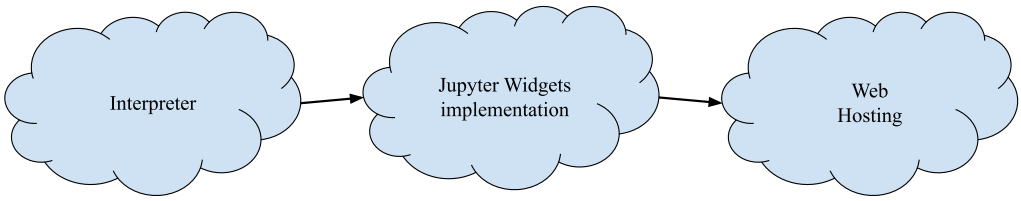

Figure 1: Dependency of sections in the tutorial

I. Interpreter

The user will walk through the implementation of the datapath and controller. To attempt to emulate the hardware as closely as possible, all paths will be computed. This is the ”meat” of the tutorial, the completion of which will convey the primary knowledge objectives as specified in the ”proposed solution” section.

Tasks:

- Control Unit
- Other things
  - ALU
  - Registers
  - Data/Instruction Memory

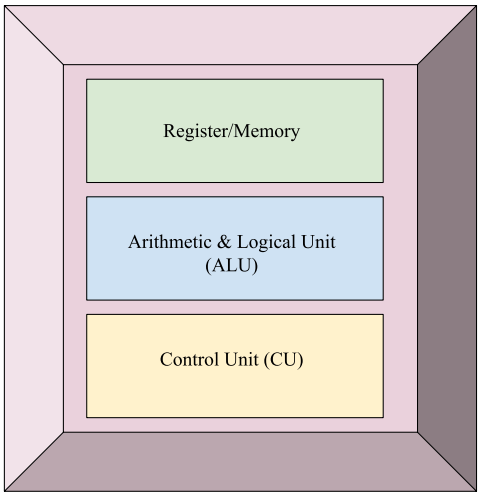

Figure 2: Architecture components

II. Jupyter Widgets implementation

Jupyter Widgets (as ipywidgets) provides a set of interactables integrated directly into a Jupyter notebook, including sliders, buttons, numerical/string inputs, and display outputs. These could be mapped to different input features (such as a foward step or command), and outputs (visually representing the registers). Upon completion users will know how to create an interactive user interface in their Jupyter notebooks.

Tasks:

- Implement program and input widgets
- Create machine command widgets (run/stop/step)
- Visualize registers with widgets

III. Web hosting

Using Voila as an interface, the Jupyter notebook with widgets can be hosted as a web app. By default all code cells are hidden, so minimal changes will be necessary to get an effective web app. This tutorial will not include deployment, but doing so follows naturally with Voila. This is especially useful as a gentle introduction to web technology for users with less experience in the subject.

Tasks: These are some tasks the team with do.

- Import and configure Voila
- Host the application to a local IP address
- Check the website and test implementation

4 Schedule

Table 1 describes the expected completion dates for each of the tutorial parts shown in Figure 1. Further subtasks can be found in Section 2.

Date	Description	Task Lead
October 23	Complete Project Plan	Henry
October 30	Outline Computer Architecture	Henry
November 6	Needs Assessment and Related Work	Hasana
November 14	Implement interpreter code	Henry
November 15	Code Review	Alex
November 20	Widget and Heb-hosting Completion	Alex
December 4	Final Presentation	Hasana
December 6	Code freeze (only writing/editing)	Hasana
December 14	Final Submissions	Alex

Table 1: Schedule

5 Contingencies

- If machine instructions are generally too complicated to implement them all, we can implement a subset which does not contain the instructions related to function calling. Better yet, left as "an exercise for the reader."
- Attempts to emphasize the hardware aspect using Python may at times be clunky, as it is not a hardware description language. In these cases, comments giving a more direct explanation will suffice.
- If the interpreter is taking too long to implement, we can drop the web app component of the tutorial, and just focus on the interpreter.