CMPT 276

Assignment 1

Two rivals on an island...

Deadline: October 5th, 2016
Last revision: September 22, 2016

Karol Swietlicki
Simon Fraser University

1 Two rivals on an island...

The year is 2222. Years of global warming and sea level rise have changed the geopolitical landscape of the world forever.

Two rival countries are dangerously crammed together on a small piece of land called Burnaby Mountain Island. No one remembers what either “Burnaby” or “Mountain” refer to, but the name stuck. The first country is called “The United Classes of South Simon-Fraseria.” The other one is “The Free Principality of North FICtonia”. The cultural divide between the two countries is a big problem. The tension is constant and unpleasant.

South Simon-Fraseria is a strange place. Many years of fighting for scarce resources that are available in those post-apocalyptic years have changed the society. There is a great need to maintain a degree of safety from bandits and their lethal ambushes. People have banded into small groups called “teams”. A collection of those teams is called a “class” and is led by an “instructor”. Strange customs indeed. However hard to understand they may be, those customs did let the country survive some very harsh times.

North FICtonia is no less strange. People from there don’t speak a common language between any of them, but their power comes from the ruins that their cities are built upon. Once upon a time there was a “university” in the area. Its remains are buried deeply now, but the North keeps on excavating deeper and deeper. Their hope is to find some ancient technology to use in warfare. Usually the things that they unearth are benign, but this is not always true.

The situation on the island is dire. North FICtonia has excavated a strange device called a “desktop computer”. Simon-Fraseria’s spies say that the device can be used to compute trajectories of upset avians launched via slingshots into herds of pigs. This is worrying. What if it could be re-purposed for more nefarious tasks?

One of the instructors of South Simon-Fraseria has requested your presence. You, just like many other people currently on the island, are a mercenary guest to South Simon-Fraseria, looking to earn enough knowledge for your travels to what you hope will become your new home. As a skilled person that has seen much of the world outside of Burnaby Mountain Island, you have been tasked to develop a desktop computer that would rival the one that North FICtonia has. The strange culture requires that everything is done as a team, so you had to quickly choose a group of people to work with. You usually work alone, but that whole “team” thing worked well for the locals... Maybe it will work for you too?

The grand library has been unsealed for your benefit. Bring your team and your quantum core laptops. You are starting by writing the heart of the machine, the processor. While the technology is lost, some documentation from an obsolete age still remains. The papyrus is remarkably well-preserved. From this you can piece together at least a simulator of a computer.

Whatever you do, please hurry! There are at most two weeks before the North figures out what to do with their device. You must have some kind of deterrent or the South will be surely defeated.

2 Seriously now...

For every piece of homework I create there is a story. You can skip section 1 entirely. I'm not gonna stop writing them, but you don't have to read them. People have reported to like them, so I keep doing it.

Your task is to write a CPU simulator for the KCPSM6 machine, as described in the KCPSM6_User_Guide_30Sept14.pdf document located at: http://tinyurl.com/h48hora

Ken Chapman's Programmable State Machine 6 is a 8-bit microcontroller, usually used inside Field-Programmmable Gate Arrays. It is a very elegant machine and very able given its small size. It is a register machine, featuring two banks of 16 registers. Most operations operate on at least one register.

3 Needles and haystacks

The document I linked you to is 124 pages long. This may be intimidating. I have spent a very long time choosing the most clearly written CPU specification. This one has the least amount of things you need to learn in order to write a simulator. There are pictures to show you what is going on and the writers did not assume a highly technical audience. There are simpler processors out there, but they are not described as well. There are also processors that are much more beautiful, but they are too large to simulate in two weeks, even given a well-coordinated team. KCPSM6 it is.

You don't have to read all of the specification though. There are only some pages that are relevant to your needs. We will not be handling assembler directives. We will not be dealing with interrupts, sleep states.

Page 35 describes the power-on state of the machine. You need this to initialize the machine to a known state at the start of the simulation.

Page 52 describes the syntax of the file that you will read in as the input. You don't need to handle a binary format as input, you will only have to deal with the text file given to you.

Pages 54 through 101 describe the instructions of the machine. From this range you can skip pages: 72, 75-77, 79-80, 83-86, 99-100.

4 Suggested division of labor

There is a lot of work here, to put it lightly. I was not kidding when I said the course won't be a cakewalk. If you don't divide the tasks, you won't make it on time. This is by design. Given two weeks there is no one person that can learn everything about CPU design to know everything and currently almost no one has the faintest idea on how a CPU works, the practice homework aside. Each person needs to get their component, learn all about it and write the code for it.

You will also need to start writing some components blind. Writing the assembly language parser will take time. If you hold back on working on other components until it is done, you will surely not make it on time. You have quickly agree on responsibilities and get to work.

I suggest the following division of labour:

- One person to handle reading the assembly code and handling labels.
- One person to manage the program counter, the stack and the CPU flags.
- One person to manage the operations on registers and constants.
- One person to manage inputs and outputs from the peripherals.
- One person to manage the program memory, (optionally) the stack and the scratchpad.

One of those tasks is substantially easier than the others. I suggest the team leader takes that task, as there will be enough work managing everything else.

Making a Gantt chart should be the first thing you do. Estimate timing to the best of your ability. Start all components as soon as possible, even if you have to make some guesses. You should be working on combining the parts within a week.

I estimate the task would take me (working alone and in my spare time) three days, but I know exactly what I'm doing. I expect each component to take a person about a full day of concentrated effort and a further two day period of working together towards integration.

5 Inputs

Your program will take as input a single file containing at most 2047 lines of code, each line containing a single machine instruction. Each line may be optionally prefixed with an adress label. This will be the program you are to execute. Read in the entire thing and only then start the execution. You don't need to handle comments.

Every time your program executes an "input" instruction you should read in, from the standard input and without printing anything, a number between 0 to 255. Treat that as the value to be written to the target register.

6 Outputs

Every time your program executes an "output" or "outputk" instruction you should print out, to the standard output, the port number followed by the output value. Use two hexadecimal digits for both numbers and make sure each of the export instructions prints to a fresh line. Separate the two numbers with a single space.

You will be marked solely on this output across multiple testcases that we will give you a week before the deadline.

7 Simulator considerations

KCPSM6 is meant to be run in a complete system. A simulator has no system, everything is make-believe. We are going to make some assumptions that allow us to work without an external system.

Assume that the program memory is exactly as large as the count of the instructions in the input file. Assume it is completely independent of all other memories.

Assume the scratchpad is the largest possible size, 256 bytes.

You can pretend that interrupts, sleep states, assembler directives, assembler comments and register aliases don't exist.

Make the "hwbuild" instruction load a 0 into the target register.

Stop the simulation when you execute the same unconditional jump instruction ten times in a row. This will happen when you have code of the format

label: JUMP label

8 Helpful resources

Any books on CPU design are going to be valuable, but not at all required. In particular the Patterson/Hennessy book with ISBN 9780123838728 might be of service, but everything you need to know can be learned from the KCPSM6 manual.

There also already exists a simulator for KCPSM6, called FIDEx. You can use it to check your correctness. You can get a free version from the official website. At least one person should learn how to use this other simulator.

Note for the people that have done the practice project: KCPSM6 has a different behavior on branches. The program counter goes up by one with every instruction executed that isn't a branch. On taken branches it is set to the branch target without being incremented. The stack is also a very different creature in KCPSM6, mainly used to store return addresses from function call. No data ever goes on the stack in KCPSM6.

Remember that your team can use an extension.

9 Questions so far

In this section I will put any questions that are frequently asked, as well as the questions that I anticipate. You can expect this section to be updated a few times during the lifetime of the assignment.

- NOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
  - That's not a question.

- Why?
  - I want you to learn a lot. Pressure tends to bring out both the best and the worst in people. This will give you a true experience of working in a team: When what you do actually matters and the tensions are high. People complain that the university did not prepare them for their actual work environment. Here is an example of a course where pressure is an actual factor. You actually have to divide the work, unlike in the iteration of this course that I had during my undergrad.

- Is this a joke?
  - Many things I do are jokes, but this one isn't.

- Where do we begin?
  - Triage the work to be done. Divide it. Chart it. Try to understand the machine, maybe play with the simulator that already exists to see how it behaves. Then get to work. The behaviors are actually very simple, there is just a lot of text to churn through.