Computer Architecture
Benchmark Homework + Project 1 + Data Analysis Homework
Fall 2025

In this project, each group will write a simulator framework and implement an instruction level simulator in it for the simple 12 (s12) ISA; write two benchmark kernels in s12 assembly language; explore performance and characteristics of the benchmarks; and report on your findings.

Task 1: Benchmark Homework (60 points total)

Each group will write at least two benchmark kernels to be shared as part of the s12 benchmark suite. This will be a homework grade. The benchmark kernels will perform the same task, but one will be written to optimize for space, and the other will be written to optimize for number of instructions. Here are the requirements for the benchmark task (e.g. what the code does):

- There must be a loop
- The input must be able to be of variable size (e.g. It operators on a list of *n* numbers). You may require that the variable size is specified as part of the input data.
- The instructions must not change based on the input size (e.g. no hard coding)
- It must fit in the limited memory space available in the s12 architecture
- It must be written in s12 assembly language.

TO SUBMIT: You will turn in two things.

1) A plain text file containing
    - A short description of your benchmark and how to use it. Begin each line of this description with "//"
    - The author(s) of the benchmark (begin line with "//") and their class section
    - The human readable assembly code (e.g. LOAD 0A)
    - Give your benchmark a short, descriptive name with a ".s12" extension.
2) A "ready to run" memory image given in the memFile file format described below with your benchmark code and example data.

Each group will have one submission. Use the following submit command:

```
submit sarahfrost compArch benchHW
```

A few examples of benchmarks are as follows:

- **Maxfinder_genSize:** Given a list of *n* numbers stored in addresses 0xFF-(*n* + 1) to 0xFF -1, and the value *n* stored in 0xFF, find the maximum of the *n* numbers and store the result in address 0xFF-(n+2).
- **Multiply_2Int:** Given two integers stored in 0xFD and 0xFE, multiply them together and store the result in 0xFF.
- **Sort_fixedSize:** Given three integers stored in address 0xFC through 0xFE, sort the numbers from smallest to largest and store the sorted list in addresses 0xFC through 0xFE.
- **Sort_genSize:** Given a list of *n* numbers stored in addresses 0xFF-(*n* + 1) to 0xFF -1, and the value *n* stored in 0xFF, sort the *n* numbers from smallest to largest. Store the final sorted list in address 0xFF-(*n* + 1) to 0xFF -1.

Task 2: Project 1 - Simulator Framework and Instruction Level Simulator (100 pts)

Using the starter code provided, you will write a simulator framework that will allow you to execute benchmarks (such as those written for the benchmark homework) and collect information about their execution. You are being given the following code:

- S12_IL_Interface.java - interface specifying required methods for the instruction level simulator

You may add any helper classes, methods, etc. that you desire. You will turn in two files:

- S12_IL.java that implements S12_IL_Interface.java
- S12_Sim.java that meets the specifications below.

S12_Sim.java Specifications

USAGE:

S12_Sim.java is the user interface for the simulator that runs from the command line and will be invoked as follows:

$> java S12_Sim <memFile> <optional: -o outputFileBaseName> <optional: -c cyclesToExecute>

memFile = name of plain text file storing state of the memory prior to beginning execution

You are welcome to add additional optional arguments and/or flags!
An example of the command line execution in practice:
$> java S12_Sim maxFinder1

This would result in the creation of two files: maxFinder1_memOut, maxFinder1_trace

CONSOLE OUTPUT:

The default output of the simulation run will display the number of cycles executed, the ending value stored in the program counter (PC), and the final value stored in the accumulator (ACC). e.g.
Cycles Executed: 2141
PC: 0xAA
ACC: 0xA51

FILE FORMAT: memFile (also the format for saving the memory out to <baseName_memOut>)

The format of the memFile is as follows. The first line consists of an 8-bit value (value of the program counter), a single space, and a 12-bit value (value of the accumulator). Each consecutive line will consist of: a two bit hex number specifying the address (00 through FF), a single space, a 12-bit value (value stored at that address). The first instruction should be stored in address 0x00, meaning the program counter will start at 0x00. (Having the PC and ACC explicitly stated allows checkpoints to be stored and restarted.)

e.g.
00000000 000000000000
00 101111010001
01 111100000000
…

FF 000000000000

Addresses not used should still be listed with a value of zero.

FILE FORMAT: <baseName>_trace
Each line will consist of a single instruction that was *executed*. Store using the assembly level description with the word form of the instruction followed by two hex digits specifying the address involved in the instruction (e.g. “LOAD 00” rather than 0100 00000000).

TODO:
S12_Sim.java will do the following:

- Validate command line arguments
- Read in and parse the specified memory file and populate the simulator’s memory
- Simulate machine execution either until it reaches the HALT command or until the specified number of cycles have been executed, whichever occurs first.
- Write to file the state of the memory after execution (baseFile_memOut) as described above.
- Write to file the instruction trace (baseFile_trace)
- Output the data to the console as described above

TO SUBMIT: Remember to submit all files needed for the simulator to compile and run (including interfaces)

- S12_Sim.java
- S12_IL.java
- S12_IL_Interface.java (though you will not modify this file)
- Any other files needed for simulator to compile and run

TASK 3: DATA ANALYSIS HOMEWORK (60 points)

Your team will write a report discussing each of the following for the pair of benchmarks your group wrote and at least two more pairs of benchmarks written by your peers. Include in your report:

- CPI for each of the benchmarks. (Spoiler alert: This is not interesting for this version of the architecture)
- Determine the largest problem each benchmark version can execute on (remember the instructions take up part of the memory!), and describe how the problem size is determined (sometimes this is straightforward, but not always)
- Create a graph comparing the number of instructions executed for each benchmark compared to the size of the list (e.g. size of the list on the x-axis, instruction count of the simulation the y-axis)
- Create a graph comparing the Number of Unique memory locations accessed compared to the size of the list.
- DISCUSS your results. What can be seen in the graphs? Is there other information or comparisons that would be useful? How would you get the data needed to support that exploration?

TO SUBMIT:

- Your report as a pdf. Please include names of all authors and what sections each author is in.
- The memFiles used to gather your data.

Use the following submit command:

```
submit sarahfrost compArch s12ILdata
```