Operating Systems

CMPT 424 • Fall 2021

*i*Project Two - 100 points

Goals

To build on the functionality from *i*Project One (all of which is required) by adding the ability to load and execute **one** user program in 256 bytes of memory as specified by the 6502a op codes documented on our class web site.

Functional Requirements

- ☐ Modify the load command to copy the 6502a machine language op codes into main memory. [10 points]
  - Put the code at location $0000 in memory
  - assign a Process ID (PID)
  - create a Process Control Block (PCB)
  - return the PID to the console and display it.
- ☐ Add a shell command, run <pid>, to run a program already in memory. [3 points]
  Note: the user should be able to execute many load/run cycles in sequence.
- ☐ Execute the running program (including displaying memory, CPU, and PCB status as well as any output). Be sure to synchronize the CPU execution cycles with clock ticks. [30 points]
- ☐ As the programs executes, update and display the memory, PCB, and CPU status (program counter, instruction reg, accumulator, X reg, Y reg, Z flag) in real time. [5 points]
- ☐ Implement line-wrap in the CLI. (This is not longer optional.) [2 points]
- ☐ [challenge] Provide the ability to single-step execution (via GUI buttons). [+8 points]
- ☐ [challenge] Allow the user to break the current program with Ctrl-C. [+4 points]

Implementation Requirements

- ☐ Develop a PCB prototype and implement it in the client **OS**. [5 points]
- ☐ Develop a memory **manager** and implement it in the client **OS**. [10 points]
- ☐ Develop a core memory prototype and implement it in the **host** OS. [10 points]
- ☐ Develop a memory **accessor** prototype and implement it in the **host** OS. [10 points]
- ☐ Develop a CPU prototype and implement it in the **host** OS. [15 points]

- ☐ Your code *must separate structure from presentation*, be professionally formatted, use and demonstrate best practices, and generally make me proud to be your teacher. [−∞ if not]
- ☐ Continue to preserve GLaDOS. There is still a neurotoxin canister next to what looks like an incinerated morality core. (Is that important?)

General Hints

Consider refactoring the host OS code into at a few parts: events, interrupts, and services.

Implement the host core memory and CPU as a separate objects and separate source code files within the host directory.

Run all accesses to memory through your host memory accessor so it can do address translations on the next project.

Read chapter 4 up to “Paging” in our text. It presents a nice look at main memory that you might find helpful.

Remember the utility of comments and how much their presence and quality affect my opinion of your work. Seriously. Bad code style will cost you a ton of points.

Specific Hints

- Continue to keep .js and .ts files in separate directories.
- The shutdown functionality must operate properly even when a program is running.
- The memory and CPU displays must update in real time. The PCB needs to update only on scheduling events, which in this project are only when execution begins and ends.
- Remember that you must continue to process interrupts while the CPU is executing.
- Always separate logic and structure from presentation. **Do not** put GUI logic in your CPU or memory routines. Keep the GUI stuff in control.ts or something similar.
- As I warned you in class, it's easy to mix up base 10 and base 16. Double check all of that.
- You are almost guaranteed to make an off-by-one error implementing or executing the BNE operation. Embrace it. Then fix it.
- Never use magic numbers. Use constants for everything other than 0, 1, and ∞.

Very Specific Hints

Follow the CPU example when implementing memory.

In the main directory:

```
Declare globally...
// Hardware (host)
var _CPU: TSOS.CPU;
var _Memory: TSOS.Memory;
var _MemoryAccessor: TSOS.MemoryAccessor;
// Software (OS)
var _MemoryManager: any = null;
```

In the host/ directory, put:

```
cpu.ts
memory.ts
memoryAccessor.ts
```

In the hardware control code :

```
_CPU = new Cpu();
_CPU.init();
_Memory = new Memory();
_Memory.init();
_MemoryAccessor = new MemoryAccessor();
```

In the OS/ directory, put :

```
memoryManager.ts
```

In the kernel bootstrap code:

```
_MemoryManager = new MemoryManager();
```

Submitting Your Work

Update GitHub with your current code and let me know that it's ready for me to grade (and which branch to grade).