Operating Systems
CMPT 424 · Fall 2017

—*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.   **[25 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.   **[10 points]**

☐ Implement line-wrap in the CLI. (This is not longer optional.)   **[2 points]**

☐ **Optional**: Provide the ability to single-step execution.   **[+10 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.   **[15 points]**
☐ Develop a core memory prototype and implement it in the **host** OS.   **[15 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 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 OS core memory and CPU as a separate objects and separate source code files within the host OS directory.

Run all accesses to memory through your client OS memory manager so it can do address translations.

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 if you're using TypeScript. • The shutdown functionality must operate properly even when a program is running. • The memory, CPU, and PCB displays must update in real time. • 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. • As I warned you in class, it's easy to mix up base 10 and has 16. Double check all of that. • You are almost guaranteed to make an off-by-one error implementing or executing the BNE operation. • Never use magic numbers. Use constants for everything other than 0, 1, and ∞.
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).

```
>load
Process ID: 0
>run 0
>12
```

Log

Sun, Aug 28th 2016, 1:33:05 pm	900
OS Idle	
Sun, Aug 28th 2016, 1:32:43 pm	474
OS CPU cycle	

Processes   `Round Robin`

PID	PC	IR	ACC	X	Y	Z	Priority	State	Location
0	34	A9	79	3	2	1	32	Running	Memory

Memory

0x020	A9	4F	8D	43	00	A9	4E	8D
0x028	44	00	A9	45	8D	45	00	A9
0x030	00	8D	46	00	A2	02	A0	42
0x038	FF	00	00	00	00	00	00	00

CPU   `LDA #$4F`

PC	IR	ACC	X	Y	Z
022	A9	79	3	2	1

```
A9 03 8D 41 00 A9
01 8D 40 00 AC 40
00 A2 01 FF EE 40
00 AE 40 00 EC 41
00 D0 EF A9 44 8D
42 00 A9 4F 8D 43
00 A9 4E 8D 44 00
A9 45 8D 45 00 A9
00 8D 46 00 A2 02
A0 42 FF 00
```