# CS 4348/5348 Operating Systems -- Project 1

You will create some C files that simulate the computer hardware needed to run a basic operating system. You will also define some additional C files that direct the hardware to load instruction set programs and run them. The necessary files are as follows:

### *memory.c*

You will write a file called *memory.c* which simulates the memory of a computer. This file will contain an integer array *MEM*, along with the following functions:

| | |
|---|---|
| mem_init(int k) | Initializes the array *MEM* to an int array of size k, and prepares *memory.c* for operations |
| mem_read() | Copies the contents of MEM[MAR] to MBR |
| mem_write() | Copies the contents of MEM[MAR] to MBR |

Memory read/write is performed using data stored in the CPU registers (MAR, MBR, etc), so there are no arguments passed to the read/write functions.

### *cpu.c*

You will write a file called *cpu.c* which simulates the CPU of a computer. This file will contain a struct called *REGS*, along with the following functions:

| | |
|---|---|
| cpu_fetch_instruction() | Reads an instruction from memory using the address stored in PC, and places the instruction into IR0 and IR1 |
| cpu_execute_instruction() | Perform the simulated system action as specified by IR0 and IR1 |
| int cpu_mem_address(int m-addr) | Returns the true memory address by computing BASE + m-addr |
| cpu_operation() | Enter an infinite loop that executes instructions until the 'exit' opcode occurs |

The REGS struct stores the registers of the CPU. The required registers are:

| | |
|---|---|
| BASE | Base Register: Stores the memory address of the currently executing program (Multiple programs may be stored in memory, so BASE is required to differentiate them) |
| PC | Program Counter: Stores memory address (relative to BASE) of the currently executing instruction |
| IR0 | Instruction Register 0: Stores the opcode of currently executing instruction |
| IR1 | Instruction Register 1: Stores the opcode of currently executing instruction |
| AC | Accumulator: Stores data for actions which accumulate data (i.e. addition) |
| MAR | Memory Address Register: Stores memory address of item being read from memory |
| MBR | Memory Buffer Register: Stores data being read from memory so it can be used |

### *load.c*

You will write a file *load.c* which loads instruction set programs into memory. The file will contain the following functions:

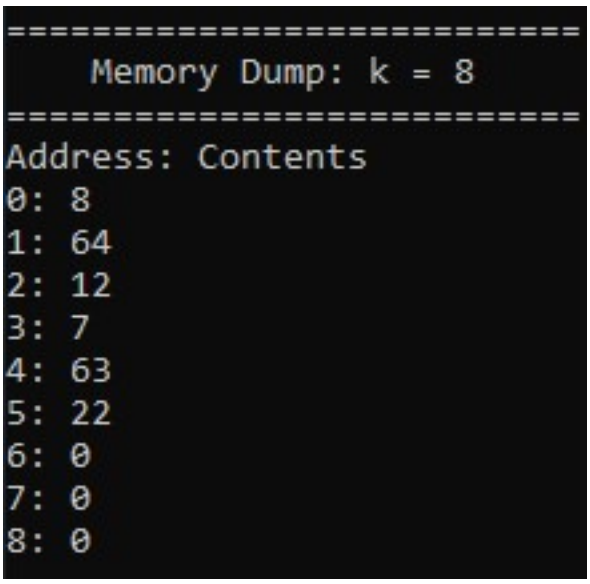
| | |
|---|---|
| load_prog(char *fname, int p-addr) | Opens a file with the filename "fname", then copies the program that is stored in the file into memory starting at the address given by p-addr |
| load_finish(File *f) | Clean up and close the file f |

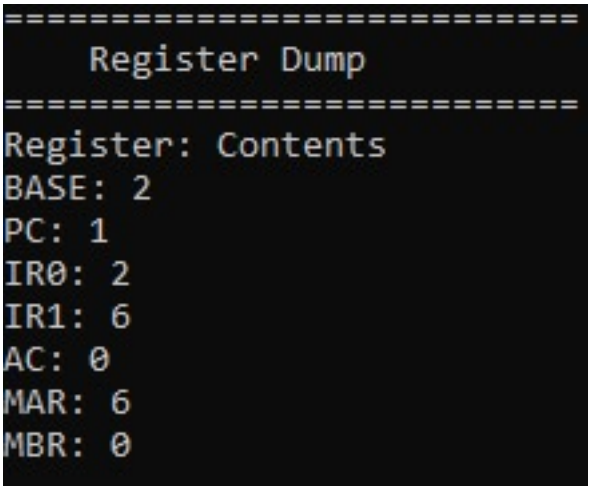
### *shell.c*

You will write a file called *shell.c* which provides a user interface for the simulated hardware. The file will contain the following functions:

| | |
|---|---|
| shell_init() | Initialize the shell module and prepare for operations |
| shell_print_registers() | Print out all CPU registers onto the command line |
| shell_print_memory() | Print out the contents of all the entire memory |
| shell_command(int cmd) | Takes the shell action specified by cmd |

We expect the memory and the registers to be printed to the command line in a very specific way. The outputs are expected to be printed as follows:




### *computer.c*

You will write a file called *computer.c* which uses all the previous files to create a functioning system. The file will contain a struct called *PCB*, along with the following functions:

| | |
|---|---|
| main(int argc, char *argv[]) | The main function and the entry point for the simulation |
| boot_system(int k) | Performs initialization of the hardware |
| process_init_PCB() | Create a PCB for an instruction set program and initialize it |
| process_set_registers() | Copy the PCB context data to the registers |

The *PCB* struct stores the registers of the CPU as well as a PID (process ID) number.

## Requirements

Your C files will need to work together to simulate an operating system that can run an instruction set program. When the program is compiled and run, the following sequence of tasks should occur:

1. Memory and CPU are initialized. Memory size should be read from "config.sys".
2. The user is prompted to enter the filename of an instruction set program, followed by a space, followed by an integer which gives the starting memory address the program will be copied to. For example:

   Enter filename and start address: prog1 16
3. The instruction set program file is loaded into memory.
4. A PCB is created for the loaded program which contains information about the program.
5. Some data in the PCB is copied to the CPU registers.
6. The CPU should begin execution of the loaded instruction set.
7. Once the CPU is complete exit the operating system.

## Project Submission

You need to submit your program **before** midnight (11:59pm) of the due date listed on eLearning. Your submission should be a .ZIP file which contains the following:

- All source code files making up your solutions to this assignment.
  - cpu.c
  - memory.c
  - load.c
  - shell.c
  - computer.c
  - Any .h header files you create.
- makefile and execute script
- A "readme" file that describes what each .c file does, and how to run the program.
  - If you did not finish the project, you need to specify which part(s) you have completed and which part(s) are missing.
  - If you submit an unfinished project without this specification, you will not receive partial credit.

Failure to follow these submission requirements will result in an automatic -10 point penalty.