CS-4348 Project 1

This project will involve you creating C files that simulate the hardware of a simplified computer, as well as some basic firmware that allows you to run programs on the computer. The programs will be written using a very limited instruction set that mimics assembly language.

Remember, ***your program must run on the UTD Giant Server!*** Any programs which cannot compile or run on the Giant Server will suffer from an automatic -50 point penalty. Instructions for logging into the Giant Server are given on the eLearning page.

You will be given a set of functions you are required to implement for each computer component. You are allowed to define additional helper functions if you wish, and you are allowed to modify what arguments are passed into each function. However, you should do your best to stick with the general architecture that has been presented in this document.

Instruction Set Programs

Your simulated computer will run programs that have been written using a very limited instruction set. The programs are text files made up of these instructions. They will be loaded by the computer, translated into their integer OP Codes, and then stored in memory with their arguments. Each instruction will consist of a command, and will potentially be followed by a space and a single argument and then a newline character. If no argument is needed for the command, then it will be immediately followed by a newline.

The possible commands and their arguments are as follows:

Command	OP Code	Argument	Description
exit	0	None	Signals the end of the current program
load_const	1	int	Load the integer given by int into the AC register
move_from_mbr	2	None	Move the integer in the MBR register into the AC register
move_from_mar	3	None	Move the integer in the MAR register into the AC register
move_to_mbr	4	None	Move the integer in the AC register into the MBR register
move_to_mar	5	None	Move the integer of the AC register into the MAR register
load_at_addr	6	None	Load an integer in memory using the address in the MAR register into the MBR register
write_at_addr	7	None	Write the integer in the MBR register to memory at the address in the MAR register
add	8	None	Add the contents of the MBR register to the AC register
multiply	9	None	Multiply the contents of the MBR register by the AC register and store the result in the AC register.
and	10	None	Perform the logical AND operation between the MBR

			register and the AC register. 0 is False, all other numbers are True. Store the result in the AC register.
or	11	None	Perform the logical OR operation between the MBR register and the AC register. 0 is False, all other numbers are True. Store the result in the AC register.
ifgo	12	addr	If the contents of the AC register is not 0, then load the integer (addr - 1) into the PC register. Otherwise, do nothing.
sleep	13	None	Do nothing.
//	None	None	Comment. Ignore this line during loading.

An example program will be provided on eLearning, however the program used to test your project will be altered to prevent students from hardcoding behavior into their projects.

memory.c

You will create a C file called memory.c which will simulate the physical memory of the computer. The physical memory will be represented by a 2 dimensional integer array. The first dimension will be of size 1024, and the second dimension will be of size 2.

You will define the following functions in the memory.c file:

Function	**Description**
int* mem_read(int addr)	Read from memory at location addr and return the data
void mem_write(int addr, int* data)	Write the given data into memory at the location given by addr.

When a program is loaded into memory, the OP Code will be loaded into position 0 at a particular address location and its argument (if applicable) will be loaded into position 1.

cpu.c

You will create a C file called cpu.c which will simulate the CPU of the computer. The cpu.c file will have a series of integers to represent the registers. The registers necessary will be:

Register	**Description**
Base	Base Register: Stores the memory address of the currently executing program.
PC	Program Counter: Stores the memory address (relative to Base) of the currently executing instruction.
IR0	Instruction Register 0: Stores the OP Code of the currently executing instruction.
IR1	Instruction Register 1: Stores the argument of the currently executing instruction, if applicable. Set to 0 if there is no argument.
AC	Accumulator Register: Stores the results of instructions
MAR	Memory Address Register: Stores memory address that will be accessed during a read/write command to memory
MBR	Memory Buffer Register: Stores the data that will be read/written to memory.

You will also define the following functions in the cpu.c file:

Function	Description
void fetch_instruction(int addr)	Read the instruction in memory at location addr, and place the OP Code in IR0 and the argument (if applicable) in IR1.
void execute_instruction()	Execute the instruction in the IR0 and IR1 registers.
int mem_address(int l_addr)	Returns true memory address by computing Base + l_addr
int clock_cycle()	Iterate through one clock cycle of the CPU, i.e. fetch an instruction and execute the command. Returns 0 when the exit command is encountered, returns 1 otherwise.

disk.c

You will create a file called disk.c which will be used to load programs into the OS memory as well as translate them into the integer OP Codes before they are stored. This file will contain the following functions:

Function	Description
void load_prog(char *fname, int addr)	Load the program with the name fname, translate it into integer OP Codes, and then store it in memory at address addr
int* translate(char *instruction)	Translate the given instruction into its associated integer OP Code. Return the argument too if applicable.

main.c

You will create a file called main.c which will be used to start the computer and drive the CPU clock cycles. You are free to define whatever functions are necessary to initialize everything.

You can hard-code file locations and system configurations here. When you load the program into memory, place it at address 4. Once the program is finished executing, print out the first 20 locations in memory.

Submission Requirements

Your submission must meet the following requirements:

- Your code will need to run a given instruction set program once, and then exit gracefully while printing the first 20 memory location.
- Your code will need to successfully compile and run on the UTD Giant Server.
- The following items must submitted in a zip file on eLearning
    - cpu.c
    - memory.c
    - disk.c
    - main.c
    - A readme.txt file that describes your code and how to run it.