ISA Project

In this project we implemented an SIMP processor simulator (resemble to MIPS processor but simpler) in C language.

First, we created a program(asm.exe) that translates an assembler code to machine code and writes it to the main memory (txt file in this program).

The second program(sim.exe) simulates the instructions from the main memory.

Assembler(asm.exe):

Input arguments: receives 2 arguments in command line application (no including "asm.exe").

1. program.asm- the assembler code
2. mem.txt – an empty file

output: writes to mem.txt the new memory following the given assembler code.

Functionality:

The program passes twice on the assembler code. In The first pass we save all the labels in the assembler code on a table with the number of line it's placed(PC address), and we translate all the rest of the code that don't uses the label's PC address to a machine code. At the end of the first pass, the program prints the memory to a temporary file named "write.txt". In the second pass, the program reads from "write.txt", replace the label's names with their PC address and prints it to "mem.txt".

Structures:

typedef struct Label label;

struct Label {

char name[MAX\_lEN\_NAME];//field name - contains label's name

int adress;//field address - contains the PC address the label was found

label\* next;// this field is for dealing with collision in the hashtable

};

typedef struct HashTable {

label\*\* items;// items is an array of pointers to labels

int size;// array's size is 'size' as the user decide

}HashTable;

In this program we used 'hashtable' technique for saving all the labels with a minimal time complex and minimal memory allocation.

Main functions:

PassOne()-

PassTwo()-

Simulator (sim.exe):

Input arguments: receives 11 arguments in command line application. (no including "sim.exe")

1. memin.txt- the output from asm.exe
2. diskin.txt- the initial data in the hard disk.

This program simulates SIMP processor

Functionality:

We used a 'w+' mode for 'memout.txt' and 'diskout.txt' so we can read and write simultaneously in the program to those text files. First, the program duplicates the 'memin.txt' and 'diskin.txt' to 'memout.txt' and 'diskout.txt'. than the program reading from 'memout.txt' the instructions and implement them. The program starts from PC=0 (PC is the number of line in 'memout') and the PC changes every instruction regarding to the specific instruction that was read.

Important variables:

int PC // the current PC the program is reading the instruction

int PC\_next // the next PC the program need to read from.

Main functions:

void Simulator(FILE\* Memout,FILE \*trace, FILE \*leds, FILE \*diskout,FILE \*hwregtrace, FILE \*regout, FILE \*cycles, FILE \*display, FILE \*irq2in);

-this is the major function. In this function the program reading from 'memout' the instructions in a 'while' loop, and breaks when the program encounter a 'HALT' instruction. It also handling an IRQ cases.

void Instructions\_0\_to\_13\_opcode(int R[], int opcode, int rd, int rs, int rt, int PC, int\* PC\_next);

void Instructions\_lw\_sw(int R[], int opcode, int rd, int rs, int rt, int PC, int\* PC\_next, FILE\* Memout);

void IO\_Instructions(FILE\* hwregtrace, FILE\* leds, FILE\* diskout, FILE\* memout, FILE\* display, int opcode, int R[], int IORegister[], int rd, int rs, int rt, int\* PC\_next, int clock\_cycle);

void routine\_timer(int IORregister[]);

void routine\_file(FILE \*irq, int IORregister[], int clock, int\* num);

void routine\_disk(int IORegister[], int\* timerdisk);