Project Purpose

This project aims to build a simple computer system with a CPU and memory. In this system, a program is available in memory, and the CPU executes the program line by line and ends the program at the end.

Implementation

The program is written in C language, which includes several functions as described below:

```
 91   /* CPU Initialize */
 92 > void CPU_init(int t){···
105
106   /* Memory Initialize */
107 > void Mem_init(char f[12]){···
127
128   /* READ Memory */
129 > int Read_memory(int addr, int wpd, int rpd){···
142
143   /* WRITE Memory */
144 > void writeMemory(int addr, int data, int wpd){···
155
156   /* RUN Memory */
157 > void runMemory(int wtpd, int rdpd){···
187
188   /* Fetch Data */
189 > int fetch(int wtpd, int rdpd){···
192
193   /* Instruction set */
194 > void Instruction_set(int wtpd, int rdpd){···
408
409   /* RUN CPU */
410 > void runCPU(int wtpd, int rdpd){···
438
439   /* Driver */
440 > int main(int argc, char* argv[]){···
496
```

First, I am Initialized the CPU by setting the registers to 0. Also, Get the Timer interrupted by a cmd argument from the user.

```
/* CPU Initialize */
void CPU_init(int t){
    PC = User_addr;                 // set Program counter to user address
    SP = User_stack;                // set Stack pointer to user stack
    IR = 0;                         // set No instruction
    AC = 0;                         // set Accumulator to 0
    X = 0;                          // set register to 0
    Y = 0;                          // set register to 0
    Counter = 0;                    // set number of instructions to 0
    Mode = User_mode;               // set Mode to User_mode

    /* get Timer_interrupt from User; cmd argument */
    Counter_set = t;
}
```

Same for Memory, I have Initialized the Memory and get the sample.txt from cmd argument by the user.

```
/* Memory Initialize */
void Mem_init(char f[12]){
    // load a program into Mem
    int offset = 0;

    // open text file
    FILE* fp = fopen(f, "r");

    // load the file into Memory
    while (!feof(fp)){
        char buff[LineBuffer_size] = "";
        // read each line into buffer
        fgets(buff, sizeof(buff), fp);
        // data line
        // convert char to instruction
        if (buff[0] >= '0' &&buff[0] <= '9')
            Mem[offset++] = atoi(buff);
        else if (buff[0] == '.')
            offset = atoi(&buff[1]);
    }
}
```

The two mentioned arguments are Implemented in the Driver function as below:

```
Mem_init(argv[2]);

CPU_init(t);
```

Also, I have created a function to interact with Memory, access the system addresses, and return the data from memory.

I have created single char for interacting with Memory:

r: read
w: write
E: end

```c
/* READ Memory */
int Read_memory(int addr, int wpd, int rpd){
    if (Mode == User_mode && addr >= Timer_interrupt){
        printf("Memory violation: accessing system address %d in user mode\n", addr);
        // error occur, exit
        write(wpd, "E", sizeof(char));
        exit(-1);
    }
    int tmp;
    write(wpd, "r", sizeof(char));      // signal
    write(wpd, &addr, sizeof(addr));    // address
    read(rpd, &tmp, sizeof(tmp));       // return data
    return tmp;
}
```

```c
/* WRITE Memory */
void writeMemory(int addr, int data, int wpd){
    // Memory Protection
    if (Mode == User_mode && addr >= Timer_interrupt){
        printf("Memory violation: accessing system address %d in user Mode\n", addr);
        write(wpd, "E", sizeof(char));  // End processes
        exit(-1);
    }
    write(wpd, "w", sizeof(char));      // signal
    write(wpd, &addr, sizeof(addr));    // address
    write(wpd, &data, sizeof(data));    // data
}
```

Then, I have created an Instruction set function as project description:
So, the CPU can get Instructions rules from this function.

```
/* Instruction set */
void Instruction_set(int wtpd, int rdpd){
    switch (IR) {
        // Load the value into the AC
        case Load_value:
            AC = fetch(wtpd, rdpd);
            break;

        // Load the value at the address into the AC
        case Load_addr:
        {
            int addr = fetch(wtpd, rdpd);
            AC = Read_memory(addr, wtpd, rdpd);
            break;
        }
        // Load the value from the address found in the given address into the AC
        case LoadInd_addr:
        {
            int addr = fetch(wtpd, rdpd);
            addr = Read_memory(addr, wtpd, rdpd);
            AC = Read_memory(addr, wtpd, rdpd);
            break;
        }
        // Load the value at (address+X) into the AC
        case LoadIdxX_addr:
        {
            // Calculate address
            int addr = fetch(wtpd, rdpd) + X;
            AC = Read_memory(addr, wtpd, rdpd);
            break;
        }
```

In the end, the Run CPU function can run the program by using the Instruction set function.

```
/* RUN CPU */
void runCPU(int wtpd, int rdpd){
    do {
        // if User_mode
        if (Mode == User_mode)
            Counter++;

        // fetch instruction to register
        IR = fetch(wtpd, rdpd);
        // execute it
        Instruction_set(wtpd, rdpd);

        // check timer interrupt flag
        if (Mode == User_mode && Counter == Counter_set){
            Counter = 0;                    // clear timer
            Mode = Kernel_mode;             // set Kernel_mode to access interrupt handler
            int tmp = SP;
            SP = Sys_stack;
            SP--;
            writeMemory(SP, tmp, wtpd);
            SP--;
            writeMemory(SP, PC, wtpd);
            PC = Timer_interrupt;
        }
    } while (IR != End);

    // End
    write(wtpd, "E", sizeof(char));
}
```

The Driver function (main) contains creation processes and pipes. Also, run the Memory and CPU functions.

```
/*Create pipe*/
// read pipe
int rdpd[2];
// write pipe
int wtpd[2];
// if pipe failed
if (pipe(rdpd) == -1 || pipe(wtpd) == -1)
{
    printf("pipe has been failed.\n");
    exit(1);
}

// create two process: CPU, Memory
int pid = fork();
```

Personal Experience

This was the first time I designed a straightforward system by building and recognizing memory and CPU performance, and I had a lot of fun.

One thing that I was a problem with was creating a variable inside the switch statement.

So, I didn't know that we must use {} inside the case statement if we want to create a variable inside it.
So that one consumed a lot of time to debug... :]

```
// Load the value at the address into the AC
case Load_addr:
{
    int addr = fetch(wtpd, rdpd);
    AC = Read_memory(addr, wtpd, rdpd);
    break;
}
```