```
#ifndef MACHINE_H
#define MACHINE_H 1

#include <iostream.h>
#include <string.h>
#include <stdlib.h>
#include "Memory.h"
#include "Processor.h"

int start, length;
Memory mem;
Processor p;

class Machine {

public:
    int Load_This(fstream&, fstream&, fstream&);
    void Simulate_This(fstream&, fstream&, fstream&);

private:
    int Char_To_Hex(char);
    int Text_To_Hex(char[]);

};

//Public Operations

int Machine::Load_This(fstream& main_input, fstream& outs, fstream& t_outs)
{
    //load the instructions as data into the machine's memory

    char header_buffer[13];
    main_input.getline(header_buffer, 14);

    if (header_buffer[0] != 'H')
    {
    t_outs << "\nError Code #8";
    exit(1);
    }

    char two_ch_buffer[2];
    two_ch_buffer[2] = '\0';
    two_ch_buffer[0] = header_buffer[1];
    two_ch_buffer[1] = header_buffer[2];

    p.Set_PC(Text_To_Hex(two_ch_buffer));

    two_ch_buffer[0] = header_buffer[9];
    two_ch_buffer[1] = header_buffer[10];

    int init_seg_addr = Text_To_Hex(two_ch_buffer);

    two_ch_buffer[0] = header_buffer[11];
    two_ch_buffer[1] = header_buffer[12];

    int segment_length = Text_To_Hex(two_ch_buffer);

    while (! main_input.eof())
    {
    char text_buffer[8];
    main_input.getline(text_buffer, 9);

    //Possible error check to be put in
        if (text_buffer[0] != 'T' && text_buffer[0] != '\0')
    {
        t_outs << "\nError Code #8";
        exit(1);
    }
    char addr[2];
    addr[2] = '\0';
    addr[0] = text_buffer[1];
    addr[1] = text_buffer[2];
    char data[5];
    data[5] = '\0';
    data[0] = text_buffer[3];
    data[1] = text_buffer[4];
    data[2] = text_buffer[5];
    data[3] = text_buffer[6];
    data[4] = text_buffer[7];

    mem.Load_Memory(Text_To_Hex(addr), Text_To_Hex(data));
    }

    t_outs << "\nInitial Register, PC, and Memory values:\n";
    p.Dump_Regs(t_outs);
    t_outs << '\n';
    mem.Dump_Mem(t_outs);
}

void Machine::Simulate_This(fstream& aux_input, fstream& outs, fstream& t_outs)
{
    bool HALT = false;
    int instruction_counter = 1;

    while(instruction_counter <= 200 && HALT == false)
    {
    int OP, U2, R, X, U1, S, mem_addr;

    t_outs << "\nInstruction#" << instruction_counter << '\n';
    mem_addr = p.Get_PC();  //get the mem_addr from the PC
    if (mem_addr > 255)
    {
        t_outs << "\nError Code #9\nFinal Register, PC, and Memory Values:\n";
        p.Dump_Regs(t_outs);
        t_outs << '\n';
        mem.Dump_Mem(t_outs);
        exit(1);
    }
    OP = mem.Get_OP(mem_addr);
    U2 = mem.Get_U2(mem_addr);
    R = mem.Get_R(mem_addr);
    X = mem.Get_X(mem_addr);
    U1 = mem.Get_U1(mem_addr);
    S = mem.Get_S(mem_addr);

    //make sure that both U1 & U2 = 0
    if(!(U1 == 0 && U2 == 0))
    {
        //invalid instruction, result in no-op
        t_outs << "\nError Code #1\n";
        p.No_Op();
        instruction_counter++;
    }
    else
    {
        HALT = p.Do_This(mem, OP, R, X, S, aux_input, outs, t_outs);
        instruction_counter++;
    }
    }
    if(HALT == false)// so instruction_counter >= 200
    {
    t_outs << "\nError Code #7\nFinal Register, PC, and Memory Values:\n";
    p.Dump_Regs(t_outs);
    t_outs << '\n';
    mem.Dump_Mem(t_outs);
```

```cpp
    exit(1);
    }
}

//Private Operations

int Machine::Char_To_Hex(char c)
{
    int hex_num = 0;
    if ((c >= 48) && (c <= 57))
    {
    hex_num = (c-48);
    }
    else if ((c >= 65) && (c <= 70))
    {
    hex_num = (c-55);
    }
    else if ((c >= 97) && (c <= 102))
    {
    hex_num = (c-87);
    }
    else
    {
    cerr << "Error Code #8";
    exit(1);
    }

    return hex_num;
}
int Machine::Text_To_Hex(char hex_text[])
{
    int hex_num = 0;
    int n = 0;
    int x = strlen(hex_text);
    while (n < x)
    {
    hex_num *= 16;
    hex_num += Char_To_Hex(hex_text[n]);
    n++;
    }

    return hex_num;
}

#endif
```