Lab 6 Report

1 Introduction

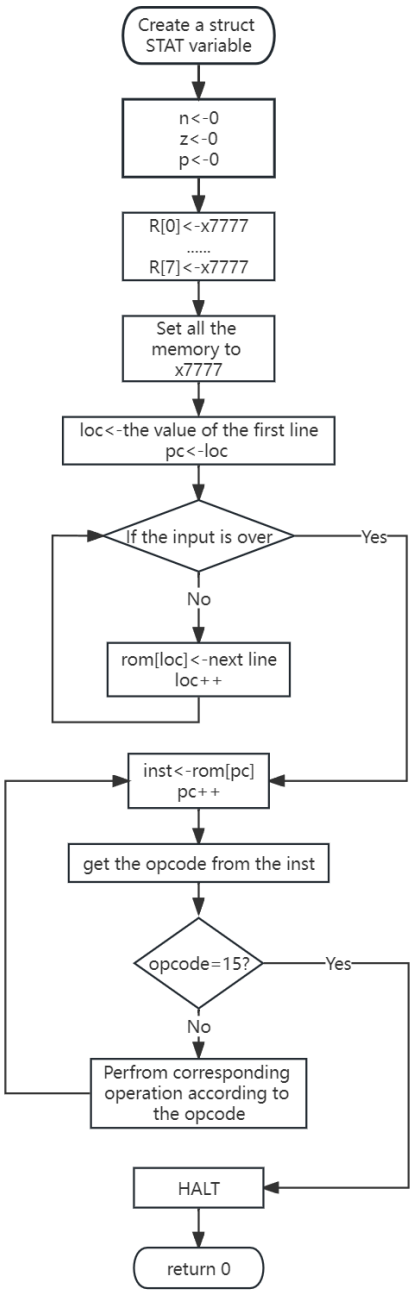
In this lab, we will write a program to execute LC-3 binary code in C. Trap routines, interrupts, exceptions, the instructions RTI, 1101, privilege mode and ACV are not required. The default values of all registers and memory locations are x7777. After the executor halts, the program will print the value of all registers.

2 Solution

First we initialize all of the registers and memory locations to x7777. And then we store all the instructions in the corresponding memory locations and set the pc. Next, we loop through the instructions and do the corresponding moves until we meet the HLAT instructions and halts.

2.1 Algorithm

The flowchart of my algorithm is as followed. The processes of initializing the registers and memories to x7777, store the instructions into the rom and execute the instructions are shown in the flowchart.



2.2 Essential part of the code

First the code of data structure definition is as followed. These two structures are often used in the program.

```
typedef struct
{
    char opcode[4];
    char value[12];
} INST;//Represents an instruction. It has fields for the opcode and value

typedef struct
{
    short int pc;
    char rom[65536][16];
    int n, z, p;
    short int R[8];
} STAT;//Holds the state of the machine, which includes flags (n, z, p), an array of registers (R), and a program counter (pc)
```

Secondly, the initialization part is being showed. Please note that while we store the instructions into the memory, we use function getchar(). When the input ends, the return value of it will be EOF. So we use this characteristic to stop the loop when we meet the end of the input.

```
void Initialize(STAT *stat)
{
//Initialize the cc
    stat->n = 0;
    stat->z = 0;
    stat->p = 0;
//Initialize the registers
    for (int i = 0; i < 8; i++)
    {
        stat->R[i] = 0x7777;
    }
//Initialize the rom
    char value[17] = "0111011101110111";
    for (int i = 0; i < 65536; i++)
    {
        CPY(stat, i, value);//Copy the string value[] to stat->rom[i]
    }
    char ins[17] = "/0";
    gets(ins);//Get the starting location
    short int loc = SEXT(16, ins);//Transform the string to its value and store it to loc
    stat->pc = loc;//Set the pc
    int ch;
    int l = 0;
//Store the instruction into the corresponding memory location until the input ends
    while((ch = getchar()) != EOF)
    {
        if(ch >= 48 && ch <= 57)
        {
            if(l == 16)
            {
                loc++;
                l = 0;
```

```
                stat->rom[(loc + 65536) % 65536][l++] = ch;
            }
            else
            {
                stat->rom[(loc + 65536) % 65536][l++] = ch;
            }
        }
    }
}
```

Thirdly, we here show the instruction parsing and execution process. Here we use a switch-case to determine which operation to execute based on the opcode.

```
int loop(STAT *stat)
{
    INST *inst = malloc(sizeof(INST) + 1);
    while (1)
    {
        PST(stat, (int)stat->pc, inst);
        stat->pc++;
        switch (USEXT(4, inst->opcode))
        {
        case 0:
        {
            BR(stat, inst);
            break;
        }
        case 1:
        {
            ADD(inst, stat);
            break;
        }
        case 2:
        {
            LD(stat, inst);
            break;
        }
        case 3:
        {
            ST(stat, inst);
            break;
        }
        case 4:
        {
            JSR(stat, inst);
            break;
        }
        case 5:
        {
            AND(stat, inst);
            break;
        }
        case 6:
        {
            LDR(stat, inst);
            break;
        }
```

```
        case 7:
        {
            STR(stat, inst);
            break;
        }
        case 9:
        {
            NOT(stat, inst);
            break;
        }
        case 10:
        {
            LDI(stat, inst);
            break;
        }
        case 11:
        {
            STI(stat, inst);
            break;
        }
        case 12:
        {
            JMP(stat, inst);
            break;
        }
        case 14:
        {
            LEA(stat, inst);
            break;
        }
        case 15:
        {
            if (USEXT(8, inst->value + 4) == 37)
            {
                TRAP(stat);
                return 0;
            }
            break;
        }
        default:
            printf("ERROR");
            break;
        }
    }
    free(inst);
}
```

The specific operations are skipped here since they have been shown in detail in the book. If you want to know in specific please read my source code.