Lab6b-Report

1 Algorithm

Global Variables as Registers and Memory

Since the word length of LC-3 is 16 bits, it's essential to create variables with data type of 16-bit length (in C, that's *short*).

- Registers include *MAR, IR, PSR, PC, R0~R7*
- Memory is replaced with an array, which size is x10000.

Initialize: Input the binary code and Store

- Initialize the R0~R7 and memory locations with 0x7777
- First scan the `ORIG.` line transforming from a string, and store it into PC
- Then scan the other lines,and store them into memory locations separatively.

LOOP infinitely until TRAP

- Use the variables to imitate the action `MAR <- PC, PC <- PC+1, MDR <- M[MAR], IR <- MDR`
- Use bitwise shift to get `bit[15:12]`, that is opcode.
- Define an enum variable and an function pointer array to identify the opcode, and enter the corresponding function.
- If the opcode is `TRAP`, output the value of R0~R7 and halt the program.

Instructions Implementation

In this executor, we ought to implement 14 intructions, which are similar to each other fundamentally. To be brief here and reserve details in code part, I want to outline the frame of functions.

- Create local variables(such as `dst, src, offset, cc`, shift bitwise after bit mask, and get information.
- Operate according to the LC-3 ISA.
- If the instruction involves offset or immediate number, sign-extension is needed(use OR bit mask).

2 Codes & Comments

```
#define BUSNUM 0x10000                  //memory access space
#define HALT 0xF025
#define MASK15_12 0xF000                //bit mask, more are omitted
enum _opcode {
    BR = 0, ADD, LD, ST, JSR, AND, LDR, STR, NOT = 9, LDI, STI, JMP,
    LEA = 14, TRAP
} opcode;                               //enum variables to identify opcodes
```

```
short MAR, IR, PSR = 1, PC;
short MEM[BUSNUM];
short R[8];
//MEM[unsigned short].
//need type conversion, otherwise pose segmentation fault!

void setCC(int dst);                    //ALU and LD* instructions set CC
short BinarytoDecimal(char *bicode);    //input binary code as string, and
                                        //convert it to signed short
void Initialize(void);
//function prototypes of instructions are omitted here
void (*fp[16])(void) = {BRfunc, ADDfunc, LDfunc, STfunc, JSRfunc, ANDfunc, LDRfunc,
STRfunc, NULL, NOTfunc, LDIfunc, STIfunc, JMPfunc, NULL, LEAfunc,TRAPfunc};
//a function pointer array to identify opcodes and call functions

short BinarytoDecimal(char *bicode) {
    short decimal = 0;
    for (int i=0;bicode[i];i++) {
        decimal <<= 1;
        decimal += bicode[i] - '0';
    }
    return decimal;
}
void Initialize(void) {
    for (int i=0;i<BUSNUM;i++) {
        MEM[i] = 0x7777;
    }
    for (int i=0;i<8;i++) {
        R[i] = 0x7777;
    }
    char bicode[17];
    scanf("%s",bicode);
    PC = BinarytoDecimal(bicode);
    unsigned short temp = PC;
    while (fscanf(stdin,"%s",bicode) == 1) { //use fscanf to test EOF
        MEM[temp++] = BinarytoDecimal(bicode);
    }
}
void ADDfunc(void) {
    int dst, src1, sign;
    signed short src2;
    //bitmask and bitwise shift to get information
    dst = (IR & MASK11_9) >> 9;
    src1 = (IR & MASK8_6) >> 6;
    sign = (IR & MASK5) >> 5;
    //bit[5]==0, src2 is register
    if (sign == 0) {
        src2 = IR & MASK2_0;
```

```
        R[dst] = R[src1] + R[src2];
    }
    //bit[5]==1, src2 is imm
    else {
        sign = IR & MASK4;
        src2 = IR & MASK4_0;
        if (sign) {
            src2 |= ~MASK4_0;//sign extension
        }
        R[dst] = R[src1] + src2;
    }
    setCC(dst);
}
//AND is very similar to ADD, thus omitted here
//NOT, LEA, LD are omitted too
void LDRfunc(void) {
    short dst, base, offset6;
    dst = (IR & MASK11_9) >> 9;
    base = (IR & MASK8_6) >> 6;
    offset6 = IR & MASK5_0;
    int sign = IR & 0x0020;
    if (sign) {
        offset6 |= ~MASK5_0;
    }
    R[dst] = MEM[(unsigned short)(R[base] + offset6)];
    setCC(dst);
}
void LDIfunc(void) {
    short dst, offset9;
    dst = (IR & MASK11_9) >> 9;
    offset9 = IR & MASK8_0;
    int sign = IR & MASK8;
    if (sign) {
        offset9 |= ~MASK8_0;
    }
    R[dst] = MEM[(unsigned short)(MEM[(unsigned short)(PC + offset9)] & 0xFFFF)];
    //attension here! Must convert signed to unsigned, otherwise cause
    //segmentation fault
    setCC(dst);
}
//ST* is very similar to LD*, thus omitted here
void BRfunc(void) {
    char n,z,p,N,Z,P;
    n = (IR & MASK11) >> 11;
    z = (IR & MASK10) >> 10;
    p = (IR & MASK9) >> 9;
    N = (PSR & 4) >> 2;
    Z = (PSR & 2) >> 1;
    P = (PSR & 1);
```

```
    if ((n & N) | (z & Z) | (p & P)) {//judge if BR is taken
        short offset9 = IR & MASK8_0;
        int sign = IR & MASK8;
        if (sign) {
            offset9 |= ~MASK8_0;//SEXT
        }
        PC += offset9;
    }
}
void JSRfunc(void) {
    R[7] = PC;
    short sign = (IR & MASK11) >> 11;
    if (sign) {//JSR
        short offset11 = IR & MASK10_0;
        sign = IR & MASK10;
        if (sign) {
            offset11 |= ~MASK10_0;
        }
        PC += offset11;
    }
    else {//JSRR
        short base = (IR & MASK8_6) >> 6;
        PC = R[base];
    }
}
void JMPfunc(void) {
    PC = R[(IR & MASK8_6) >> 6];//including RET(JMP R7)
}
void TRAPfunc(void) {
    for (int i=0;i<8;i++) {
        printf("R%d = x%04hX\n",i,R[i]);//format print
    }
    exit(0);//halt the program
}
int main() {
    Initialize();
    while (1) {
        MAR = PC++;
        IR = MEM[(unsigned short)MAR];
        opcode = (IR & MASK15_12) >> 12;
        if (fp[opcode]) fp[opcode]();
    }
    return 0;
}
```