# ALGORITHM

My lab 6b program is written in C.

First of all, an array of short with 0xFFFF spaces was arranged to simulate the memory in LC-3. I tried node to store the instructions but eventually found out that it`s much more complicated than array and it seems acceptable for to “wasted” at most 65kb memory in computer today. Then I set unsigned short REG[7], CC, PC and CurrentIS (current instruction) to simulate the registers I need. Rest thing to do now is to translate operations in finite-state-machine into C language.

Before we start executing, read the very first instruction considered as .ORIG. Read instructions left and load them one by one into our simulated memory. Program will stop when and only when CurrentIS is a TRAP instruction.

# MAIN CODE

Code here shows the main executor program. It is basically the C-language version of FSM in appendix, except several unrequired functions such as privilege and TRAP.

while (1) {

CurrentIS = MEM[PC];

PC++;

OpCode = CurrentIS >> 12;//decoder`

switch (OpCode) {

case 0b0000://BR

……

break;

case 0b0001://ADD

……

case 0b1111://TRAP, NOT REQUIRED

for (int i = 0; i < 8; i++) {

printf("R%d = x%04hX\n", i, REG[i]);

}

return 0;

}

}

Code here shows how I load instructions into simulated memory. Unsigned short MEM[0xFFFF] is a global variable.

//load instructions into MEM

unsigned short Load()

{

char temp;

char\* IS;

unsigned short NowAddr = 0;

unsigned short Head = 0;

//read a 16bits string in buffer

//return address of string after read 16 chars

//or return NULL while reading EOF

IS = MyGetLine();

Head = StrToNum(IS);

NowAddr = Head;

while ((IS = MyGetLine()) != NULL) {

MEM[NowAddr] = StrToNum(IS);

NowAddr++;

}

return Head;

}

Code here shows part of tool functions I wrote to simplify my executor. It should be noticed that a lot of bitwise operation are used to conveniently separate instructions.

//turn a 16bits string into an unsigned short

unsigned short StrToNum(char\* str)

{

unsigned short factor = 1;

unsigned short result = 0;

for (int i = 15; i >= 0; i--) {

result += (str[i] - 0x30) \* factor;

factor \*= 2;

}

return result;

}

//pick up imm in an instruction, len is the length of imm you want

short SEXT(unsigned int instruction, int len)

{

short result = 0;

switch (len) {

case 5:

result = instruction << 11;

result = result >> 11;

if (result & 0b10000) {

result = result | 0b1111111111100000;

}

break;

case 6:

result = instruction << 10;

result = result >> 10;

if (result & 0b100000) {

result = result | 0b1111111111000000;

}

break;

case 9:

result = instruction << 7;

result = result >> 7;

if (result & 0b100000000) {

result = result | 0b1111111000000000;

}

break;

case 11:

result = instruction << 5;

result = result >> 5;

if (result & 0b10000000000) {

result = result | 0b1111100000000000;

}

}

return result;

}

//return a regester number in a particular place in an instruction

unsigned short TellReg(unsigned int instruction, int highbit)

{

unsigned short result = 0;

switch (highbit) {

case 11:

result = instruction << 4;

result = result >> 13;

break;

case 8:

result = instruction << 7;

result = result >> 13;

break;

case 2:

result = instruction << 13;

result = result >> 13;

break;

}

return result;

}

# MISTAKES I HAVE MADE

It`s a slopy mistake I`ve made at the beginning that I forgot to initialize all memory and registers as 0x7777.

LEA instruction will not affect CC and I need to use “%04hX” in printf() to make sure the answer is output in right format. These two mistakes was solved with the help of my classmates in our DingTalk group. Thanks to them.

I accidentally lost a 0 in immediate number 0b10000000000 in JSR executor. It costed me a bunch of time to find it out actually because it`s unexpected for me and I am in lack of machine codes to test my program. Thanks to CSDN for providing me code examples.