Derek McCreery & Hennok Tadesse

N00771054 & 823-393-335

CENG 356

April 20, 2017

Lab 10

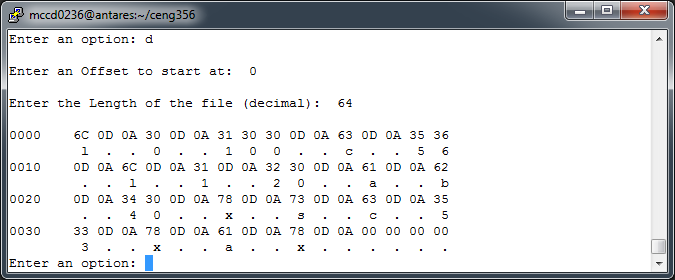
**Description**

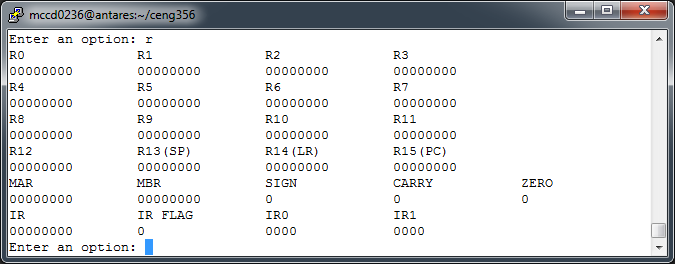
The goal of the assignment was to create our own virtual CPU. This includes creating and managing our own registers, flags, and functions. The CPU should be able to load an instruction set into memory and run it like a program. These instruction sets would alter values in registers and allow the user to display the values said registers.

**Final State of the Program**

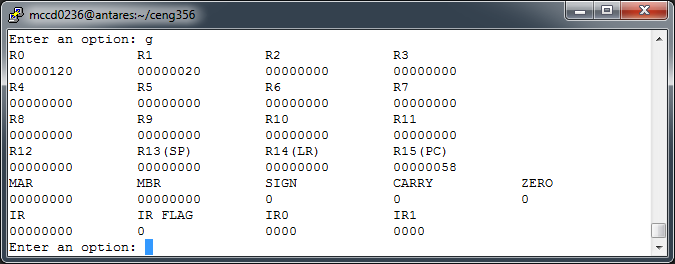
Currently the program is able to perform all of the required functions set out by the assignments. I also was able to implement a version of the Trace function that works independently from the Fetch function as I was unable to implement that.

**Sample Output**

Memory Dump  


Register Display  


Displayed registers after running through a program



**Testing**

|  |  |  |
| --- | --- | --- |
| **Test** | **Test Step** | **Actual Results** |
| LoadFile Function:  This test was performed to ensure that a file would be loaded into memory and display the correct file size. | >Pressing ‘L’ or ‘l’ that would execute the function.  >Entering the filename that would be loaded into memory.  >Entering a filename that did not exist. | >After entering the filename to be loaded into memory, the file size is displayed in decimal and as well as in hex.  >Entering a filename that did not exist had returned an error message saying “The file does not exist”. |
| MemDump Function:  This test was performed to ensure that specified amount of bytes are being displayed and that the memory dump is displaying at the proper offset as well as entering the length and offset which is greater than the memory. | >Pressing ‘D’ or ‘d’ that would execute the function.  >Entering the length of bytes to be displayed in hex.  >Entering the offset which specifies the part to be displayed. | >Entering the length and offset which is greater than the memory will result in an error message saying “out of bounds”  >Entering correct values for the length and offset will display the memory dump at the specified values. |
| MemMod Function:  This test was performed to ensure that the existing value will get replaced with the entered hex value and that the function will exit if the user enters “.”. | >Pressing ‘M’ or ‘m’ that would execute the function.  >Entering the offset in which to start modifying  >Entering the new hex value to replace the value at the entered offset  >Entering “.” To end the function | >The existing value at the given offset gets replaced by the new value entered and then increments the address.  >Entering a “.” Terminates the function and brings up the menu. |
| WriteFile Function:  This test was performed to ensure that the file was created correctly with the specified size. | >Pressing ‘W’ or ‘w’ that would execute the function.  >Entering the filename  >Entering the file size | >Successfully created the file with the specified filename and file size. |
| displayRegisters Function:  This test was performed to ensure that the registers and flags are displayed according to their current state. | >Pressing ‘R’ or ‘r’ that would execute the function. | >The registers and flags are displayed successfully according to their current state. |
| zeroRegisters Function:  This test was performed to ensure that the registers and flags would reset to 0 when executed. | >Pressing ‘Z’ or ‘z’ that would execute the function. | >Cleared all registers and flags |
| go Function:  This test was performed to ensure that the function would go through the instruction file and execute the instructions accordingly. | >Loading in a file that has the contents of (l 0 100 l 1 100 a s c 21 s u 33 x a x)  Which would load 100 to register 0, load 100 to register 1, add register 0 and 1, sub register 0 and 1, conditional branch to location 21, sub register 0 and 1, unconditional branch to 33 and then add register 0 and 1 and finally terminates the go function.  >Pressing ‘G’ or ‘g’ that would execute the function. | >Successfully executed load instruction, add from register 0 and 1, sub from register 0 and 1, conditional branch to a location and unconditional branch to a location. |

Currently the program can only read in very specifically made programs and that would be a cause of many errors or even segmentation faults. Other than this the program works correctly and I was unable to find any errors. To test that each implemented function was working in both the Go and Trace functions, I ran a program that used all of them. This file is included in the .zip file and is named “test.txt”.

**Conclusions**

In conclusion, this assignment was used to show us and think critically about how a CPU would work form a logical point of view. I believe that the assignment was a good approach to further teach and reinforce the some of the ideas presented in the course. From here, to further the project, additional functions should be added as well as making some of the existing code more flexible, for example having the inputs more than one character long.

**Appendix**

/\*

\* Name: Derek McCreery & Hennok Tadesse

\* File name: virtualCpu.c

\* Last revision: April 18, 2017

\*/

#include <stdio.h>

#include <stdlib.h>

#define MEMORY\_MAX 0x3E80

long int regs[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

//13 Stack pointer

//14 Link Register

//15 Program Counter

int sign=0; //1 = negative

int zero=0; //1 = value is zero 0 = value is a number that is not zero

int carry=0; //1=carry

long int mbr=0;

long int mar=0;

short int ir0=0;

short int ir1=0;

int irf=0;

int ir=0;

int stopFlag=0;

int main()

{

int quit=1;

char input;

char newLineHolder;

char \* buffer;

buffer = malloc(sizeof(char)\*MEMORY\_MAX);

int loadResult=0;

int dumpOffset = 0;

int dumpLength = 0;

int modOffset = 0;

printf("d Dump memory\n");

printf("g Run the entire program\n");

printf("l Load a file into memory\n");

printf("m Memory modify\n");

printf("q Quit\n");

printf("r Display registers\n");

printf("t Trace - execute one instruction\n");

printf("w Write file\n");

printf("z Reset all registers to zero\n");

printf("?,h Display a list of commands\n");

while(quit==1)

{

printf("Enter an option: ");

scanf("%c", &input);

switch(input)

{

case 'd':

case 'D':

printf("\nEnter an Offset to start at: ");

scanf("%d", &dumpOffset);

printf("\nEnter the Length of the file (decimal): ");

scanf("%d", &dumpLength);

printf("\n");

MemDump(buffer, dumpOffset, dumpLength);

break;

case 'g':

case 'G':

Go(buffer);

break;

case 'l':

case 'L':

loadResult=LoadFile(buffer,16384);

if(loadResult == -1)

{

printf("The file does not exist\n");

}

if(loadResult != -1)

{

printf("Size of file: %u (0x%04x in hex)\n", loadResult, loadResult);

}

break;

case 'm':

case 'M':

printf("Enter the offset in memory (decimal): ");

scanf("%d",&modOffset);

printf("\n");

MemMod(buffer, modOffset);

break;

case 'q':

case 'Q':

quit=0;

break;

case 'r':

case 'R':

displayRegisters();

break;

case 't':

case 'T':

Trace(buffer);

break;

case 'w':

case 'W':

WriteFile(buffer);

break;

case 'z':

case 'Z':

resetRegisters();

printf("Registers Reset\n");

break;

//Done

case '?':

case 'h':

case 'H':

printf("d Dump memory\n");

printf("g Run the entire program\n");

printf("l Load a file into memory\n");

printf("m Memory modify\n");

printf("q Quit\n");

printf("r Display registers\n");

printf("t Trace - execute one instruction\n");

printf("w Write file\n");

printf("z Reset all registers to zero\n");

printf("?,h Display a list of commands\n");

break;

}

//temporary fix to input not being taken in on main menu.

while(input != '\n'){

fscanf(stdin, "%c", &input);

}

}

return 0;

}

/\*

Resets the registers and flags to their default setting of 0

\*/

void resetRegisters()

{

int x;

for(x=0;x<16;x++)

{

regs[x]=0;

}

sign=0;

zero=0;

carry=0;

mbr=0;

mar=0;

ir0=0;

ir1=0;

irf=0;

ir=0;

stopFlag=0;

}

/\*

Loads a file into memory

\*/

int LoadFile(void \*memory, unsigned int max)

{

FILE \*file;

char filename[100];

int bytesRead=0;

int count=0;

printf("Enter the name of the file: ");

scanf("%s", filename);

file=fopen(filename,"r");

if(file != NULL)

{

//go to the end of the file, get the position of it, then go to the start, this is to see if the file is truncated.

fseek(file,0L,SEEK\_END);

count=ftell(file);

fseek(file, 0L,SEEK\_SET);

bytesRead=fread(memory,1,max,file);

fclose(file);

if(count>max)

{

printf("The file was truncated\n");

}

else

{

printf("The File was not truncated\n");

}

}

else

{

return(-1);

}

return bytesRead;

}

/\*

Creates an empty file with a user inputted name

\*/

void WriteFile(void \* memory)

{

char filename[50];

int size = 0;

int loop;

printf("Enter a filename: ");

scanf("%s", filename);

printf("Enter the size for the file: ");

scanf("%d", &size);

FILE \*file;

file = fopen(filename, "w");

for(loop = 0; loop < size; loop++)

{

fprintf(file, "\0");

}

fwrite(memory, 1, size, file);

fclose(file);

printf("File Created Successfully\n");

}

/\*

Dumps the current state of memory, allowing the user to specify the offset to start at and how much of the memory to display from the offset

\*/

void MemDump(void \*memptr, unsigned offset, unsigned length)

{

if (offset >= 16384 || offset <0)

{

printf("Invalid offset, setting to default offset 0.");

offset=0;

}

//getting the position to end at

unsigned int end = offset + length;

int loop,loop2,loop3;

//loop until the end of specified data

for (loop = offset; loop < end; loop += 16)

{

//Prints the current location in memory

if (loop != offset)

putchar('\n');

{

printf("%04X\t", loop);

}

for (loop2 = loop; loop2 < (loop + 16); loop2++)

{

//If the length is reached before the end of the line exit loop

if (loop2 == (length + offset))

{

printf("\n\t");

for (loop3 = loop; loop3 <= (length + offset - 1); loop3++)

{

if (isprint(((char\*)memptr)[loop3])) {

printf(" %c ", ((char\*)memptr)[loop3]);

}

else {

printf(" . ");

}

}

break;

}

//Display the Hex value

printf("%02X ", ((unsigned char\*)memptr)[loop2]);

//Print the ASCII value or a . if unable to

if (loop2 == (loop + 16) - 1)

{

printf("\n\t");

for (loop3 = loop; loop3 < (loop + 16); loop3++)

{

if (isprint(((char\*)memptr)[loop3])) {

printf(" %c ", ((char\*)memptr)[loop3]);

}

else {

printf(" . ");

}

}

}

}

}

printf("\n");

}

/\*

Allows the user to directly modify the memory.

\*/

void MemMod(void \*memptr, int offset)

{

//ensure offset is set within boundaries

if (offset >= 16384 || offset <0)

{

printf("Invalid offset, setting to default offset 0.");

offset=0;

}

char mminput[sizeof(int)];

unsigned int value;

while (1)

{

//Displays info about current location

printf("Current address\t0x%04X\n", offset);

printf("Value in memory:\t(%02X)\n", ((unsigned char\*)memptr)[offset]);

printf("Enter a 2 digit hex number or . to exit: ");

fgets(mminput, sizeof(int), stdin);

//checks if input is a . to leave

if (strcmp(mminput, ".\n") == 0) {

printf("Exiting modify\n");

break;

}

//checks if values entered is valid

//always prints out the first time, unsure why

else if (!isxdigit(mminput[0]) || !isxdigit(mminput[1])){

printf("Enter a valid 2 digit hex number.\n\n");

continue;

}

//if the new value is valid, place it in memory and move to next location

else{

sscanf(mminput, "%X", &value);

printf("New value is: %02X\n\n", value);

((char\*)memptr)[offset] = value;

offset++;

}

}

}

/\*iscarry()- determine if carry is generated by addition: op1+op2+C

C can only have value of 1 or 0.\*/

int iscarry(unsigned long op1,unsigned long op2, unsigned C){

if ((op2== 16385)&&(C==1))

return(1); // special case where op2 is at MAX32 (16385)

return((op1 > (16385 - op2 - C))?1:0);

}

/\*

Runs the program loaded into memory

\*/

void Go(void \*memptr)

{

char input;

char\* mem=memptr;

int reg;

char svalue[100];

int value;

regs[15]=0;//reset program counter to 0 to read through memory from start

stopFlag=0;

int x;

int tester=0;

int y=0;

int conBranch=0;

while(stopFlag==0)

{

conBranch=0;

conBranch=conFlag();

input=((char\*)mem[regs[15]]);

if(input=='x' || input == NULL || input=='\0')//stop

{

stopFlag=1;

}

/\*

conditional branch

condition set to always be true

target to jump to is n-1

where n is the location in memory in decimal

\*/

if((input=='c') && (conBranch==1))

{

regs[15]+=3;

//clear svalue of any extra values

for(y=0;y<100;y++)

{

svalue[y]= '\0';

}

//take in new PC

x=0;

//loop while the next char is not a new line character and append it to the string that will become the inputted value

while( ((int)((char\*)mem[regs[15]])) != 13 )

{

svalue[x++]=((char)((char\*)mem[regs[15]++]));

}

value = atoi(svalue);

//set PC to new value

regs[15]=value;

}

/\*

unconditional branch

target to jump to is n-1

where n is the location in memory in decimal

\*/

if(input=='b')

{

regs[15]+=3;

//clear svalue of any extra values

for(y=0;y<100;y++)

{

svalue[y]= '\0';

}

//take in new PC

x=0;

//loop while the next char is not a new line character and append it to the string that will become in inputted value

while( ((int)((char\*)mem[regs[15]])) != 13 )

{

svalue[x++]=((char)((char\*)mem[regs[15]++]));

}

value = atoi(svalue);

//set PC to new value

regs[15]=value;

}

if(input=='l')//load

{

regs[15]+=3;

//converts the char number to an int number, ,atoi() was giving me an error and this was the work around I came up with

reg=((int)((char\*)mem[regs[15]]))-48;

regs[15]+=3;

x=0;

for(y=0;y<100;y++)

{

svalue[y]= '\0';

}

//loop while the next char is not a new line character and append it to the string that will become in inputted value

while( ((int)((char\*)mem[regs[15]])) != 13 )

{

svalue[x++]=((char)((char\*)mem[regs[15]++]));

}

value = atoi(svalue);

loadRegister(value,reg);

}

if(input=='a')//add regs 0 and 1

{

addRegisters();

}

if (input=='s')//subs reg 1 from reg 0

{

subRegisters();

}

regs[15]++;

}

displayRegisters();

}

/\*

Runs the program loaded into memory, displaying the registers after every character read

\*/

void Trace(void \*memptr)

{

char input;

char\* mem=memptr;

int reg;

char svalue[100];

int value;

regs[15]=0;//reset program counter to 0 to read through memory from start every time

stopFlag=0;

int x;

int tester=0;

int y=0;

int conBranch=0;

getchar();

while(stopFlag==0)

{

conBranch=0;

conBranch=conFlag();

input=((char\*)mem[regs[15]]);

printf("Line read: %c\n", input);

if(input=='x' || input == NULL || input=='\0')//stop

{

stopFlag=1;

}

/\*

conditional branch

condition set to always be true

target to jump to is n-1

where n is the location in memory in decimal

\*/

if((input=='c') && (conBranch==1))

{

regs[15]+=3;

//clear svalue of any extra values

for(y=0;y<100;y++)

{

svalue[y]= '\0';

}

//take in new PC

x=0;

//loop while the next char is not a new line character and append it to the string that will become the inputted value

while( ((int)((char\*)mem[regs[15]])) != 13 )

{

svalue[x++]=((char)((char\*)mem[regs[15]++]));

}

value = atoi(svalue);

//set PC to new value

regs[15]=value;

}

/\*

unconditional branch

target to jump to is n-1

where n is the location in memory in decimal

\*/

if(input=='b')

{

regs[15]+=3;

//clear svalue of any extra values

for(y=0;y<100;y++)

{

svalue[y]= '\0';

}

//take in new PC

x=0;

//loop while the next char is not a new line character and append it to the string that will become the inputted value

while( ((int)((char\*)mem[regs[15]])) != 13 )

{

svalue[x++]=((char)((char\*)mem[regs[15]++]));

}

value = atoi(svalue);

//set PC to new value

regs[15]=value;

}

if(input=='l')//load

{

regs[15]+=3;

//converts the char number to an int number, ,atoi() was giving me an error and this was the work around I came up with

reg=((int)((char\*)mem[regs[15]]))-48;

regs[15]+=3;

x=0;

for(y=0;y<100;y++)

{

svalue[y]= '\0';

}

//loop while the next char is not a new line character and append it to the string that will become in inputted value

while( ((int)((char\*)mem[regs[15]])) != 13 )

{

svalue[x++]=((char)((char\*)mem[regs[15]++]));

}

value = atoi(svalue);

loadRegister(value,reg);

}

if(input=='a')//add regs 0 and 1

{

addRegisters();

}

if (input=='s')//subs reg 1 from reg 0

{

subRegisters();

}

regs[15]++;

displayRegisters();

printf("\nPress Enter to execute next line\n");

getchar();

}

}

/\*

check flags for special cases

\*/

int conFlag()

{

if (regs[1]==20)

return 1;

else

return 0;

}

/\*

load a value into a register

\*/

void loadRegister(long int value, int loc)

{

regs[loc]=value;

}

/\*

add registers 0 and 1 and place result in reg 0

\*/

void addRegisters()

{

regs[0] += regs[1];

carry=iscarry(regs[0],regs[1], carry);

if (regs[0]==0)

zero=1;

else

zero=0;

if(regs[0]<0)

sign=1;

else

sign=0;

}

/\*

sub register 1 from 0, result in register 0

\*/

void subRegisters()

{

regs[0] -= regs[1];

if (regs[0]==0)

zero=1;

else

zero=0;

if(regs[0]<0)

sign=1;

else

sign=0;

}

/\*

Displays the Registers

\*/

void displayRegisters()

{

printf("R0\t\tR1\t\tR2\t\tR3\n");

printf("%08d\t%08d\t%08d\t%08d\n",regs[0],regs[1],regs[2],regs[3]);

printf("R4\t\tR5\t\tR6\t\tR7\n");

printf("%08d\t%08d\t%08d\t%08d\n",regs[4],regs[5],regs[6],regs[7]);

printf("R8\t\tR9\t\tR10\t\tR11\n");

printf("%08d\t%08d\t%08d\t%08d\n",regs[8],regs[9],regs[10],regs[11]);

printf("R12\t\tR13(SP)\t\tR14(LR)\t\tR15(PC)\n");

printf("%08d\t%08d\t%08d\t%08d\n",regs[12],regs[13],regs[14],regs[15]);

printf("MAR\t\tMBR\t\tSIGN\t\tCARRY\t\tZERO\n");

printf("%08d\t%08d\t%01d\t\t%01d\t\t%01d\n",mbr,mar,sign,carry,zero);

printf("IR\t\tIR FLAG\t\tIR0\t\tIR1\n");

printf("%08d\t%01d\t\t%04d\t\t%04d\n",ir,irf,ir0,ir1);

}